MATEMATICKO-FYZIKÁLNÍ
FAKULTA
Univerzita Karlova

BAKALÁŘSKÁ PRÁCE

Matúš Maďar

Nutriční asistent

Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Michal Kopecký, Ph.D.

Studijní program: Informatika

Studijní obor: Obecná informatika

Praha 2021

Ďakujem RNDr. Michalovi Kopeckému, Ph.D. za jeho neoceniteľné rady pri pomoci s písaním tejto práce a navádzaním na správnu cestu pri vývoji aplikácie. Taktiež sa chcem veľmi pekne poďakovať Janke Bátoryovej za jej veľmi cennú pomoc pri korektúre práce a Adamovi Hornáčkovi za jeho nesmierne dôležité rady s technologickými otázkami ohľadom Javy. Takisto sa chcem poďakovať všetkým kamarátom ktorí mi pomohli pri tejto práci, či už s jej vypracovaním alebo vnuknutím dobrého nápadu na tému.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V …..... dne..... podpis

Název práce: Nutriční asistent

Autor: Matúš Maďar

Katedra / Ústav: Katedra softwarového inženýrství

Vedoucí bakalářské práce: RNDr. Michal Kopecký, Ph.D.

Abstrakt: Práca sa zaoberá skúmaním a implementovaním adaptívneho generovania jedálnička do mobilnej aplikácie pre platformu Android. Aplikácia pomôže používateľovi dosiahnuť cielenú hmotnosť nastavením jeho jedálnička presne na mieru, pričom dovoľuje vychýliť sa od stanoveného plánu. V prípade vychýlenia a skonzumovania nenavrhovaného jedla aplikácia dokáže reagovať v reálnom čase a adaptívne toto vychýlenie vyriešiť podľa voľby používateľa. Aplikácia taktiež zohľadňuje stravovanie v reštauráciách a implementuje možnosť vyhľadávania a zápisu reštauračných jedál z okolia užívateľa. Návrh a implementácia aplikácie umožňuje jej jednoduché rozšírenie vďaka modulárneho návrhu. UI a modul ktorý generuje jedálničky fungujú oddelene, preto môžeme jednoducho meniť UI s ktorým pracuje používateľ, bez toho, aby to spôsobilo nefunkčnosť aplikácie. Modul generujúci jedálničky je rovnako modulárny a je možné použiť v aplikácii vlastné algoritmy na generovanie jedálničkov alebo iné dáta než sú použité v tejto práci.

Klíčová slova: Android, Výživa, Heuristika

Title: Nutrition assistant

Author: Matúš Maďar

Department: Department of Software Engineering

Supervisor: RNDr. Michal Kopecký, Ph.D.

Abstract: This thesis explores and implements the options for adaptive generation of meal plans. We implement our approach as a mobile application for the Android operating system. The application aims to help the user to achieve the desired weight loss or weight gain results by custom made recommendations and, at the same time, allowing small deviations from the meal plan. In case of this deviation from the proposed meal plan, the application adapts to such cases with respect to user preference. The additional feature of the application is built-in support for exploring the meal options in nearby restaurants. Our application design is modular. The implementation of the user interface is separate from the logic behind the meal plan generation. The module for meal plan generation can also be easily extended to support new meal plan generation approaches. Also, the data we currently use can be easily replaced by any other data source for food information.

Keywords: Android, Nutrition, Heuristic

Obsah

1. Úvod

„Jste to, co jíte“.

Stravovanie je dôležitý, no častokrát zanedbávaný aspekt v našom živote. S rozvojom technológií a konzumizmu sa ľudská lenivosť dostala do extrémnych hraníc, čo napomohlo rozvoju civilizačných ochorení. Viacero štúdií spája početné množstvo civilizačných chorôb práve s našim moderným životným štýlom[1].

Spoločne s pravidelným pohybom a cvičením je správne stravovanie základom zdravého životného štýlu a dôležitý faktor nášho zdravia. Športovci niekedy používajú frázu: „Ani ten najkrajší kvet bez kvalitnej pôdy nevyrastie“. Tým nechceme povedať, že sa nemáme hýbať (čo je nedielnou súčasťou zdravého života), že stačí sa len správne stravovať a všetky choroby nás začnú zázračne obchádzať. Práve naopak, chceme tým poukázať na to, že nestačí sa len hýbať, ale je veľmi dôležité dbať aj na to, čo konzumujeme.

Či už je niekto vrcholový atlét, profesionálny vzpierač, človek trpiaci alergiou alebo intoleranciou na nejakú potravinu alebo úplne obyčajný človek ktorý si ide sem tam zabehať, pre každého môže byť užitočné udržiavať si prehľad o prijatej strave a tým sa uistiť o dostatočnom príjme potrebných živín. Ako si však taký prehľad robiť? Využijeme potenciál tam, kvôli čomu sme sa vôbec do týchto civilizačných problémov dostali. Využijeme moderné technológie, ktoré nosíme každý jeden z nás vždy pri sebe vo vlastnom vrecku.

1.1 Cieľ práce

Momentálne s dostupnými aplikáciami na trhu, ktoré rozoberieme v kapitole Konkurencia, je správa prijatej potravy relatívne zložitý proces. Ideálne by bolo, keby nám stačilo iba vyfotiť to, čo máme na tanieri a aplikácia spoločne so vstavanou umelou inteligenciou sa postará o všetku matematiku a zaznamenanie jedla. Aj keď takéto pokusy už vznikajú (časť s nápadmi v Modul A1 – Nutričný manažér), sú z veľkej časti nedokonalé a nefungujú dostatočne presne. Preto ak máme záujem o zaznamenávanie prijatej potravy, buď si musíme jedlo a suroviny

k nemu potrebné vážiť a manuálne zapisovať do aplikácie, alebo budeme musieť dodržiavať statické jedálničky[1] dané pre celý deň. V prípade, že by sme si však chceli dať niečo iné než čo máme zadané v jedálničku, tak už musíme vedieť čím dané jedlo môžeme nahradiť, čo sa nutričných hodnôt týka. Toto je však demotivujúci faktor pre veľa začiatočníkov a ľudí, ktorí by aj chceli dodržiavať správne stravovanie, no nechcú sa tomu vyslovene aktívne venovať. Zvyčajne to dopadne tak, že človek preto po takejto aplikácii ani nesiahne, alebo ju po krátkom čase prestane používať, pretože je to veľká dávka energie, ktorú do toho musí investovať.

Cieľom tejto práce bude vyvinutie proof-of-concept[2] mobilnej aplikácie, ktorá umožní používateľovi jednoduché a intuitívne monitorovanie prijatej potravy vo väčšine situáciách v ktorých sa vyskytne keď sa bude stravovať. Namiesto zdĺhavého zapisovania a skúmania čo zjesť, chceme vytvoriť aplikáciu, ktorá čo najviac odbremení používateľa od vyššie spomínaných problémov a namotivovať ho k spravovaniu prijatej potravy. Dôraz budeme klásť na automatické ad-hoc [3] navrhovanie jedál tak, aby užívateľ vedel čo má jesť a dodržal svoje kalorické a nutričné limity. Pointou tohto inovatívneho prístupu je aby užívateľ nemusel striktne dodržiavať dopredu daný jedálniček.

Základné kategórie formy stravovania by sme mohli rozdeliť nasledovne:

1. Stravovanie polotovarmi či potravinami zakúpenými z obchodu (potraviny, supermarket...) bez špeciálnej úpravy (varenie, pečenie...).
2. Stravovanie doma urobeným jedlom, tj. niečo navarené, napečené... V podstate čokoľvek čo si človek spraví doma, veľakrát aj podľa nejakého receptu.
3. Stravovanie jedlom v reštauráciách (tento výraz budeme ďalej v práci používať na zahrnutie reštaurácií, kaviarní, krčiem, fastfoodov, jedální,

[1] Myslíme tým to, že musíme dodržiavať dopredu vytvorené jedálničky z receptov, ktoré nám dohromady dávajú potrebné nutričné hodnoty

[2] https://en.wikipedia.org/wiki/Proof_of_concept

[3] Pod pojmom ad-hoc v tomto kontexte myslíme navrhovanie jedálnička šitého na mieru podľa parametrov a upravovaného v reálnom čase tak, aby spĺňal nutričné potreby užívateľa

menz a iných stravovacích zariadení ponúkajúcich jedlo do jedného súhrnného pojmu) alebo tzv. take-out jedlo z reštaurácie (primárne fastfoody).

Vytvorením aplikácie, ktorá rieši zaznamenávanie nutričných hodnôt a kalórií vo všetkých troch spomínaných kategóriách, spoločne s ad-hoc generovaním jedál túto aplikáciu odlíši od existujúceho množstva dostupných produktov. Tie sa buď venujú iba menovaným kategóriám jednotlivo, alebo ak aj implementujú všetky 3 kategórie, tak sa musíme striktne držať dopredu nagenerovaných jedálničkov ak chceme dodržať svoje limity.

1.2 Základné delenie

Celá aplikácia by sa dala rozdeliť na 4 časti:

1. Nutričný manager **(A1)** – táto časť slúži na zaznamenávanie prijatých živín a ich spravovanie
2. Receptár **(A2)** – táto časť obsahuje recepty s pomocou ktorých si vie užívateľ uvariť na mieru šité jedlá
3. Reštaurácie **(A3)** – v tejto časti užívateľ nájde reštaurácie a ich ponuky spoločne s nutričnými hodnotami jednotlivých jedál
4. Guidance bot **(A4)** – tento modul bude slúžiť ako kvázi-AI, ktorý bude mať na starosti matematické výpočty nutričných hodnôt užívateľa na základe vstupných parametrov a adaptívne navrhovanie jedál prispôsobené k zadaným dátam z už skonzumovaných potravín užívateľom počas dňa

Za hlavné časti budeme v práci považovať moduly **(A1)** a **(A4)**. Najväčší dôraz bude kladený na modul **(A4)**, keďže to je niečo, čo chceme spraviť ako novú funkcionalitu. **(A2)** a **(A3)** budú chápané skôr ako prototypy, ktoré nebudeme v práci implementovať do takej hĺbky, aby sme ich mohli považovať za plnohodnotné moduly. Napriek ich dôležitosti a vhodnosti vo finálnom produkte určeného na predaj, pre naše akademické účely a demonštráciu modulu **(A4)** bude stačiť ich implementácia v obmedzenej forme. V sekcii Architektúra však ukážeme, že dizajn tejto aplikácie bude navrhnutý tak, aby bol modulárny a aby šlo tieto 2 moduly,

rovnako ako aj prípadné iné s inou funkcionalitou, v budúcnosti plnohodnotne reimplementovať. Detailnejší popis jednotlivých častí bude potom v kapitole Špecifikácia.

2. Analýza

V tejto kapitole si rozoberieme a zanalyzujeme existujúce produkty na trhu. Na základe analýzy vyvodíme záver, čo vlastne bude naša práca implementovať, aby sme vytvorili aplikáciu, ktorá je v niečom nová a inovatívna. Po analýze produktov bude nasledovať analýza dát. Keďže robíme aplikáciu, s pomocou ktorej si chceme monitorovať nutričné hodnoty a kalórie zo zjedených potravín, je dôležité mať k tomu dostatok správnych dát. Preto venujeme aj tejto problematike hĺbkovú analýzu a osobitnú sekciu.

2.1 Konkurencia

Na trhu existujú kvantá aplikácií so zameraním na zdravý životný štýl, správne stravovanie či pravidelné a korektné cvičenie. Či už si otvoríme *App Store* na *iOS* alebo *Google Play* na *Androide*, tak na obidvoch platformách už v dnešnej dobe máme na tieto aplikácie dokonca vyhradenú osobitnú kategóriu *Health & Fitness*. Pri takomto množstve ponúkaných aplikácií nás to nabáda, že je skoro až naivné si myslieť, že neexistuje riešenie pre náš problém. V skutočnosti však zdanie klame.

Ak by sme vo voľnom čase chodili cvičiť a chceli svoju snahu patrične zefektívniť, tak musíme používať viacero aplikácií naraz. Každá z týchto aplikácii pritom bude na niečo iné, a častokrát sa stane, že sú to aplikácie z konkurenčných táborov. Žiadnu synchronizáciu medzi nimi tak očakávať nejde. Toto môžeme považovať za prvý z argumentov, prečo máme vôbec motiváciu niečo vytvoriť.

Ďalší z argumentov je skutočnosť, že väčšina aplikácií pre kontrolu nutričných hodnôt predpokladajú našu znalosť správneho stravovania, pretože žiadne návrhy čo si máme dať neponúkajú. Síce to nie je problém týchto aplikácií, pretože to nie je ich primárna funkcia, no nie každý človek chce venovať svoj čas vzdelávaniu sa o nutričných hodnotách jednotlivých potravín aj keď sa chce zdravo či striedmo stravovať a mať pod kontrolou svoju výživu. Ak už aj nájdeme aplikáciu ktorá túto možnosť ponúka, teda že nám aj nageneruje jedálniček, tak sa musíme striktne držať toho, čo vygenerovala. Ak by sme si dali čo i len krajec chleba s maslom a nebolo to explicitne v jedálničku, tak by na to aplikácia nevedela v reálnom čase zareagovať a

nastaviť nám jedálniček nanovo. Naše očakávanie je, aby sme vedeli čo môžeme zjesť ďalej a krajec chleba s maslom by nebol v konečnom dôsledku nad limit.

Máme aplikácie na zadávanie nutričných hodnôt a kalórií zo zjedeného jedla alebo potravín no nevieme čo máme jesť. Máme plno receptárov a webových stránok s receptami ale je veľmi pracné si ich zapisovať. A keď sa ideme najesť niekam von do reštaurácie, tak ani nevieme kam sa máme ísť najesť, aby sme splnili svoje nutričné plány.

Posledné spomínané, teda jedenie vonku, je v našich končinách veľmi chabo integrované do akejkoľvek aplikácie ktorá rieši kontrolu prijatej potravy. Na jednej strane dôvodom môže byť nízka dostupnosť dát s ktorými sa dá pracovať, na druhej strane však existujú služby, ktoré disponujú dátami, ktoré sú len na krok od toho, aby s nimi šlo plnohodnotne pracovať v rámci integrácie ponúkaných jedál do aplikácií na kontrolu príjmu nutričných hodnôt. Pre príklad, služba **Zomato**[4] disponuje dátami o reštauráciách a ponúkaných jedlách, nedisponuje však dátami o ich nutričných hodnotách a kalóriách. Ak už však disponujem dátami o reštauráciách a ich menu, dodať tam dáta o nutričných hodnotách a kalóriách by už nemal byť tak veľký problém. Ak by tu bola snaha spolupráce medzi jednotlivými spoločnosťami, alebo by tu bol nejaký externý faktor, ktorý by ich k tomuto zlepšovaciemu kroku donútil, prípadne im s ním pomohol, tak som si pomerne istý, že by sa to stalo. Lenže ono sa to nedeje a myslím si, že ten problém tkvie niekde inde, než len v lenivosti k zmene.

Po analýze trhu si môžeme všimnúť, že náramne veľké množstvo aplikácií zaoberajúcich sa problematikou *Health & fitness* je lokalizovaná pre americký trh. To má samozrejme neblahé následky na komfort používania týchto aplikácií v našich európskych pomeroch. Viacero jedál tam nemusíme nájsť, a naopak, veľa jedál a potravín ktoré tam nájdeme sa nevyskytujú v našich európskych obchodoch. Tým pádom sa tieto aplikácie (aj keď sú pomerne kvalitné a ponúkajú dobré služby) v našich končinách v podstate nepoužívajú. A práve tento fakt, že sa u nás nenachádza tak veľa dobrých a kvalitných lokalizovaných produktov, podľa môjho názoru spôsobuje nízku mieru konkurencie na európskom trhu. Preto sa tu vyskytujú aplikácie, ktoré nie že by boli zlé, no určite by sa dali spraviť lepšie a kvalitnejšie,

[4] https://www.zomato.com

ako napríklad na spomínanom americkom trhu, ale kvôli tomu, že tu nemajú konkurenciu, tak si držia monopol a nemajú dôvod na to, aby niečo menili.

2.1.1 Kritériá hodnotenia

V tejto kapitole si načrtneme hlavné kritériá, podľa ktorých budeme hodnotiť ďalej v práci už existujúce konkurenčné produkty. Napriek tomu, že sa budeme v tejto práci snažiť implementovať hlavne moduly **(A1)** a **(A4)**, kvôli rozšíriteľnosti a objektívnej potrebe v eventuálnom finálnom predajnom produkte budeme brať ohľad aj na moduly **(A2)** a **(A3)**. Tým pádom, veľký dôraz pri porovnávaní budeme klásť práve na kvalitu spracovania a funkcionality týchto štyroch jednotlivých modulov po častiach v rámci konkurencie. Okrem toho sa pozrieme aj na lokalizáciu aplikácií a aký je ich cieľový trh. Je to informácia, ktorá nám vie napovedať viac o danom produkte. Tieto kritéria budú to hlavné o čo sa budeme v našej práci opierať. Zdôrazňujeme, že pri hodnotení a kritike konkurencie sa nebudeme zameriavať na kvalitu spracovania UI[5] alebo UX[6], pretože to nie je to, čo sa snažíme našou prácou zlepšiť, takže to pre nás nie je priorita. Našou prácou sa snažíme ukázať a vyrobiť novú zaujímavú funkcionalitu. Preto bude pre nás dôležité, ako fungujú konkurenčné produkty z obsahového a implementačného hľadiska, aby sme si z toho mohli niečo produktívne odniesť a poučiť sa, prípadne dotvoriť to, čo evidentne chýba.

V prípade analýzy produktu ***Kalorické tabuľky***[7], nakoľko autor bol dlhodobým užívateľom tejto aplikácie, budeme vychádzať aj z jeho vlastných skúseností a pocitov, nie len všeobecného stanoviska a voľne dostupných informácií na internete. Ostatné produkty boli autorom používané buď iba krátku dobu na otestovanie, alebo vôbec, a v tom prípade bude analýza pozostávať primárne z verejne dostupných informácií o aplikácii a prvého dojmu.

[5] https://en.wikipedia.org/wiki/User_interface_design

[6] https://sk.wikipedia.org/wiki/User_experience_design

[7] https://www.kaloricketabulky.sk

2.1.2 Konkurenčné produkty

2.1.2.1 Kalorické tabuľky

Kalorické tabuľky je lokálna aplikácia zameraná na náš trh. Obsahuje 4 jazyky, slovenčinu, češtinu, ukrajinčinu a ruštinu. Nájdeme tu všetko, čo potrebujeme a očakávame od nutričného manažéra **(A1)**. Nakoľko sa jedná o lokálnu aplikáciu, tak jej používanie v našom prostredí je relatívne pohodlné - ich databáza obsahuje väčšinu produktov, ktoré nájdeme v našich obchodných reťazcoch a skenovanie čiarového kódu je samozrejmosť. Okrem makroživín ponúka aplikácia aj prehľad mikroživín (vitamíny a minerály), čo môže byť pre niekoho zaujímavým bonusom. K mobilnej aplikácii má aj webovú stránku, ktorá má až na drobné detaily rovnakú funkčnosť ako aplikácia. V rámci **(A1)**, ak si odmyslíme neglobálnosť aplikácie, môžeme usúdiť, že je aplikácia prehľadná, má príjemné UI a implementuje všetky potrebné vlastnosti dobrej **(A1)**.

Aplikácia implementuje aj časť **(A2)**, no v tomto prípade sa už nejedná o akýsi nadštandardný produkt, ale skôr priemer až podpriemer.

Vieme si pozrieť komunitné recepty (pridávané ľuďmi). Pri prehľadávaní receptov máme možnosť filtrovania podľa kalórií a makroživín ako aj s pomocou kľúčových slov. Na webovej stránke je možnosť si recepty zoradiť; na mobilnej aplikácii som nič také nenašiel. Webová stránka má v tomto oproti aplikácii navrch, čo považujem za zlyhanie, pretože väčšinu svojho používania užívateľ praktizuje na mobile. Čo považujem za ďalší problém, je to, že recepty sa síce dajú hodnotiť, no užívateľ nevidí počet ohodnotení. Tým pádom, keď vidíme recept s 5 hviezdičkami, nevieme, či dostal 5 hviezdičiek od jedného človeka, alebo od 100 ľudí (čo je samozrejme s ohľadom na objektivitu veľký rozdiel). Zároveň mi tu chýba možnosť akokoľvek komentovať recepty, pretože to zbytočne komplikuje možnosť spätnej väzby a kontinuálneho vylepšovania receptov komunitou.

(A3) ako aj **(A4)** nie sú implementované vôbec.

Výhody

- Pekná prehľadná webová stránka aj mobilná aplikácia na zapisovanie kalórií
- Postačujúca databáza jedál a surovín v (pre nás) 99% prípadoch

- Skener na čiarové kódy pre polotovary z (lokálnych) obchodov funguje pomerne spoľahlivo (aj keď nie bezproblémovo)
- Obsahuje pridávanie jedál z receptov ako aj možnosť pridávať vlastné recepty
- Filtrovanie receptov
- Kontroluje aj mikronutrienty
- Free verzia nás nijak veľmi neobmedzuje oproti používaniu premium

Nevýhody

- Nemá anglickú lokalizáciu (nie je globálna)
- Rozpoloženie receptov a celkové prevedenie filtrovania si viem predstaviť krajšie
- Hodnotenie receptov síce je, ale nepíše koľko ľudí + nie je možnosť recepty okomentovať
- Neobsahuje žiadnu magickú funkciu (až na jej „monopol" CZ/SK trhu) kvôli ktorej by bola omnoho lepšia než zbytok trhu
- Absencia **(A3)** aj **(A4)**

2.1.2.2 Myfitnesspal

Myfitnesspal[8] je vedúca jednotka na trhu aplikácií so zaznamenávaním kalórií a nutričných hodnôt zjedeného jedla s viac ako 50 miliónmi stiahnutiami z *Google Play*[9]. Aplikácia je jednoznačne globálneho rázu; na výber ponúka 15 jazykov. Na ich webovej stránke sa píše, že vo svojej databáze majú cez 11 miliónov záznamov o jedlách. Len pre porovnanie, ***Kalorické tabuľky*** majú v databáze cez 200 tisíc záznamov. Po vlastnom otestovaní som našiel v ich databáze aj produkty ako Študentská pečať (čokoláda) či Horalky od Sedity, a to všetko oskenovaním čiarového kódu. Pre niekoho kto cvičí kardio cvičenia bude určite potešujúce to, že aplikácia ponúka synchronizáciu s veľkým množstvom Health zariadení (fit-náramky, smart hodinky...). Z hľadiska **(A1)** sa teda bezpochyby jedná o špičkový globálny produkt.

[8] https://www.myfitnesspal.com

[9] https://play.google.com/store/apps/details?id=com.myfitnesspal.android

(A2) v tomto produkte takisto zaostáva. Ak by sme opomenuli UX, tak ako sme si vytýčili na začiatku, tak síce môžeme tvrdiť, že aplikácia modul **(A2)** implementuje, no forma akou ju implementuje je (z pohľadu autora) tak komplikovaná a nešikovná, že je nutné to spomenúť. Nenájdeme tu žiadnu jednoduchú a rozumnú formu prehliadania receptov. Všetky recepty aké služba ***Myfitnesspal*** ponúka (či už v mobilnej aplikácii alebo na webe), nájdeme iba v blog sekcii služby. Pri prehliadaní, až na pár kľúčových slov, nemáme v podstate žiadnu možnosť filtrovania. Okrem toho, nie je vidieť žiadne hodnoty (keby sme videli aspoň kalórie a makroživiny) dokým neklikneme na jedlo[10]. Na to, že je to líder v tejto oblasti, tak modul **(A2)** nemajú perfektne zvládnutý. Jedna dobrá a šikovná vec ktorú pri module **(A2)** je nutné podotknúť je to, že aplikácia povoľuje používať recepty z iných stránok a na základe šikovného parsovania dokáže importovať recept spoločne s ingredienciami z danej stránky po zadaní URL.

Modul **(A3)** sa v ***Myfitnesspal*** objavil pomerne nedávno. Aj tento krok, keďže sa k nemu uchýlil gigant v tejto oblasti, značí, že ľudia majú eminentný záujem na tom, aby si zaznamenávali nutričné hodnoty aj pri stravovaní vonku[11]. Databáza reštaurácií je pomerne veľká; ich stránka hovorí o 500 000 reštauráciách po celom svete (nie len U.S.). Nájdeme tu aj reštaurácie a kaviarne ak skúsime lokalitu mimo U.S., napr. Praha. Databázu berú od spoločnosti ***Foursquare***[12]. Táto firma, zdá sa, že je veľkým hráčom na poli s real-world dátami lokalít a miest. Viac o dátach si však povieme v osobitnej sekcii tejto práce – Analýza – Dáta.

Čo je v tomto celom problém, tak sú už konkrétne dáta o ponúkaných jedlách a menu reštaurácií. Aplikácia síce ponúka databázu reštaurácií, no čo už konkrétne zariadenia ponúkajú sa dozvieme iba vo veľmi špecifických prípadoch. Ako užívateľ môžete akurát tak požiadať o dodanie jedálnička pre nejakú reštauráciu ktorá ho

---

[10] Ku dňu 15.10.2020 mobilná aplikácia už má funkciu Discover Recipes, kde si vieme celkom prehľadne prezerať recepty a vidíme aj kalórie pod obrázkom, takže nie je nutné rozkliknúť. Recepty sú síce kategorizované na sekcie ako Boost Immune system, Vegan, Gluten-free... no stále chýba možnosť akéhokoľvek filtrovania, čo z nášho pohľadu znamená, že reálne tam žiadnu konkurencieschopnú funkcionalitu nedodali

[11] https://blog.myfitnesspal.com/myfitnesspal-introduces-restaurant-logging/

[12] https://foursquare.com/

nemá a čakať. V konečnom dôsledku, infraštruktúru vybudovanú majú a ostáva už len na komunite a tretích stranách, kedy, ako rýchlo a za akých podmienok sa budú tie dáta postupne dopĺňať[13].

Implementácia **(A4)** nie je.

Výhody

- Veľká databáza jedál aj z obchodných reťazcov a niektorých reštaurácií
- Globálna aplikácia
- Veľké množstvo možností a funkcií (otázka však zostáva, či to výhoda pre bežného používateľa naozaj je)
- Synchronizácia s inými Health aplikáciami a zariadeniami
- Dokáže stiahnuť a vypočítať nutričné hodnoty rôznych receptov z rôznych iných stránok s pomocou URL
- Nepriama implementácia modulu **(A3)**

Nevýhody

- Aplikácia síce je globálna, chýba jej však CZ/SK jazyk.
- Zlé rozpracovanie receptov – nemá vlastnú šikovnú stránku dedikovanú receptom (iba v časti blog). Na mobilnej aplikácii je to lepšie
- Zlé rozpracovanie receptov – nemá filter (má iba nejaký pseudo, ku ktorému sa dá dostať cez veľmi veľa klikania)
- Z aplikácie ide silne cítiť speňažovanie na každom kroku. Prišiel som si predsa zapísať to čo som jedol aby som mal kontrolu nad svojimi nutričnými hodnotami a nie pozerať na blog či hľadať kde sa to jedlo vôbec zapisuje
- Free verzia je síce postačujúca pre väčšinu prípadov, no ***Myfitnesspal*** sa snaží vložiť premium všade, kde to len ide
- **(A3)** implementované nejakým spôsobom síce je, ale dáta jedál jednotlivých reštaurácií vo veľkej miere chýbajú a neexistuje žiadne ich prehliadanie v okolí pomocou mapy

[13] Ku dňu 15.10.2020 bolo autorom zistené, že funkcionalitu už z aplikácie stiahli. Prečo to tak spravili je otázne, avšak vyhľadávať menu reštaurácií je už nemožné. Firma tvrdí, že jedlá z reštaurácií sa stále nájsť dajú, len už to nejde na mape - https://support.myfitnesspal.com/hc/en-us/articles/360045757052-Retiring-Restaurant-Logging-Feature-8-1-20

2.1.2.3 Spoonacular / Edamam

Spoonacular[14] aj **Edamam**[15] sú americké riešenia spoločne aj s ich vlastnými aplikáciami, ktoré si sú vo veľa veciach podobné, tak ich rozoberieme spoločne.

Ako bolo spomínané, obe riešenia sú lokalizované pre americký trh. To znamená, že v našich pomeroch sú tieto aplikácie v podstate nevyužiteľné. To však nič neuberá na ich obecnej funkčnosti.

Obe riešenia ponúkajú svoje vlastné aplikácie, ktoré implementujú moduly **(A1)** aj **(A2)**. Modul **(A3)** v týchto riešeniach bohužiaľ nenájdeme, avšak v ich databáze jedál sa dajú nájsť aj jedlá z väčších obchodných reťazcov. Po krátkom čase však zistíme, že obe riešenia nemajú signifikantný počet užívateľov ich aplikácií a že reálne praktické využitie týchto produktov sa skrýva v ich API[16], ktoré ponúkajú, čo bude rozvinuté ďalej v kapitole Analýza – Dáta.

Okrem základných vecí ktoré ponúkajú (jedlá, recepty, potraviny, polotovary), tak ako všetky iné databázy jedál, ukážeme zopár zaujímavostí:

- Obe riešenia disponujú NLP[17], takže v prípade speech-to-text vie užívateľ v reči povedať čo zjedol a interné algoritmy rozpoznajú a zapíšu do tabuľky konkrétne potraviny
- Veľké spektrum kategórií (***Edamam*** udáva 28 druhov nutrientov a 40 rôznych diét a alergií)
- (***Edamam***) Jedlá a ich recepty sú vyhľadávané na internete a tie najrelevantnejšie sa zobrazujú užívateľovi správne naparsované s pomocou interných algoritmov priamo v aplikácii
- ***Spoonacular*** obsahuje ceny potravín v obchodoch
- Generovanie jedálničkov na základe zadanej diéty (aj keď v obmedzenej forme)
- Generovanie nákupného zoznamu na základe vygenerovaného jedálnička

---

[14] https://spoonacular.com/

[15] https://www.edamam.com/

[16] https://en.wikipedia.org/wiki/API

[17] Natural language processing - https://en.wikipedia.org/wiki/Natural_language_processing

- (***Spoonacular***) Možnosť zadať si aké suroviny mám doma a vyhľadávanie receptov bude zohľadňovať to, čo už máme a prioritizovať tak jedlá vo vyhľadávaní, ktoré obsahujú naše suroviny

2.1.2.4 Suggestic

Suggestic[18] je americké riešenie, ktoré je pravdepodobne najviac vybavené technológiami spomedzi všetkých spomínaných konkurenčných produktov. Keďže sa znovu jedná o americký produkt, tak v našich končinách jeho používanie nie je pohodlné a nemá veľký význam, nakoľko databáza neobsahuje naše produkty z obchodov. Rovnako, veľa funkcionalít ktoré táto aplikácia ponúka tak v našich pomeroch fungovať nebudú. Na území USA resp. Kanady je to však o inom.

Aplikácia implementuje **(A1)** aj **(A2)** no v obidvoch prípadoch aplikácia mierne stráca. Pri module **(A1)** vidíme ako obrovský problém nemožnosť si nastaviť kalórie a nutričné hodnoty manuálne. Toto je tak základná vec, že v produkte takého rázu by sme absenciu určite neočakávali. Pri **(A2)** bolo pri testovaní zistené, že chýba explicitné filtrovanie jedál podľa rôznych kategórií. Kategorizovanie jedál je iba na základe toho, či sa jedná o raňajky, obed, večeru alebo snack. V skutočnosti však filtrovanie jedál funguje implicitne (užívateľ si na začiatku zvolí akú diétu chce dodržiavať), preto sa dá pochopiť, že nemáme možnosť filtrovať recepty explicitne.

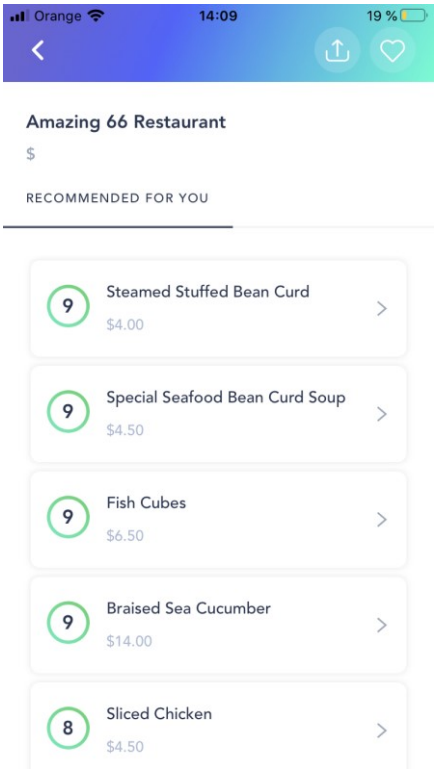
Na začiatku nás aplikácia prevedie krátkym kvízom o našich stravovacích zvyklostiach, alergiách, obmedzeniach a zároveň čo chceme v stravovaní dosiahnuť, kam sa chceme dostať a ako chceme naše stravovacie návyky nastaviť. Na základe toho nám potom aplikácia ponúkne viacero stravovacích plánov, ktoré by mali byť najbližšie k našim vyplneným odpovediam. Z nich si jeden máme potom vybrať a na základe neho nám aplikácia bude zostavovať jedálniček. V tomto momente už začína vstupovať platená aplikácia do hry, pretože veľmi veľa plánov je sprístupnených len premium užívateľom. Základné plány sú sprístupnené aj free užívateľom a myslím si, že určite to na začiatok postačí, pokým si človek vybuduje návyk.

Po zvolení plánu nám aplikácia vygeneruje jedálniček na týždeň dopredu pozostávajúci z raňajok, obeda, večere a snacku. Konkrétne jedlá si môžeme meniť

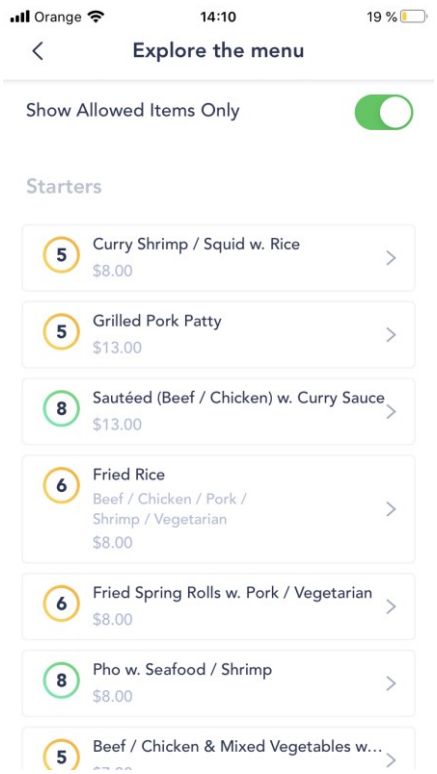
[18] https://www.suggestic.com/

ak je tam niečo čo nám nechutí. Aplikácia teda rozumne dáva človeku možnosť ako sa stravovať správne ak dodržuje jedálniček, no žiadnym spôsobom do návrhov neinkorporuje potraviny a polotovary a reštaurácie z modulu **(A3)** (ktorý aplikácia implementuje) tak, ako to chceme spraviť my v module **(A4)**.

Ako sme už avizovali, tak aplikácia implementuje modul **(A3)**. Modul je v tomto prípade implementovaný veľmi dobre. Podľa údajov z ich stránky aplikácia má dáta o zhruba 520 tisíc reštauráciách. Výhodou je to, že aplikácia iba okrem nutričných hodnôt ponúka aj cenu za akú si dané jedlo vieme kúpiť. Čo je však pre nás z technického hľadiska dôležité, tak je schopnosť aplikácie hodnotiť jedlá číslom od 0 po 10 ako veľmi sa hodia do stravovacieho programu, ktorý sme si vybrali.



Obrázok 1: Screenshot ukazujúci bodové ohodnotenie jedla z reštaurácie



Obrázok 2: Screenshot ukazujúci bodové ohodnotenie jedla z reštaurácie (2)

Problém v tomto hodnotení je ale v tom, že aplikácia dáva hodnotenie čisto len na základe kompatibility daného jedla so zvoleným plánom. Na nutričné hodnoty alebo kalórie z hľadiska dodržania limitov aplikácia ohľad neberie. Preto, keď sa niekto chce stať napr. vegetariánom a nevie čo jesť, tak pre takého človeka táto aplikácia vie byť veľkým prínosom, pretože ho núti a ukazuje mu, čo má jesť, aby si udržal správne stravovacie návyky vegetariána. No v našom prípade, kedy chceme kombinovať jedenie s udržiavaním hmotnosti, s cvičením a s chcením mať príjem potravy pod kontrolou to veru nepomôže.

Posledná vec ktorú by som ešte rád spomenul tak je funkcia, ktorú žiadna konkurenčná aplikácia neponúka a to je *Suggestic Lens*[19]. Táto funkcia je akýsi „menu skener“ s pomocou ktorého si môžeme oskenovať menu v reštaurácii a na základe AI nám vyhodnotí priamo v reálnom čase, ktoré z jedál je pre nás najlepšia voľba. Technológia je to určite super, no predpoklad je anglický jazyk, a tým pádom je to použiteľné iba v anglicky hovoriacich krajinách, teda určite nie u nás.

[19] https://www.youtube.com/watch?v=L7XW0I1U0m8

V konečnom dôsledku je táto aplikácia veľmi zaujímavým produktom na trhu z ktorého si vo viacero veciach vieme zobrať inšpiráciu, ako napríklad koncept bodového ohodnocovania jedál podľa toho, ako veľmi nám sedia zo „zdravotného“ hľadiska do nášho jedálnička. Má veľmi veľkú databázu receptov, jedál ako aj reštaurácií a používa viacero zaujímavých technológií k tomu, aby prilákala svojich zákazníkov. Táto aplikácia sa však nezameriava úplne na rovnakú cieľovú skupinu ako my. Zameriava sa skôr na správne stravovacie návyky, zvyklosti a zdravé stravovanie. To sa samozrejme nijako nevylučuje s tým, čo sa snažíme robiť my, no v našom prípade sa budeme snažiť balansovať medzi zdravým stravovaním ale zároveň korektným dodržiavaním stanovených limitov nutričných hodnôt a kalórií.

Výhody

- Veľká databáza jedál, no najmä stravovacích zariadení spoločne s dátami o ponúkaných jedlách a menu
- Modul **(A3)** zahrňuje ceny o ponúkaných jedlách
- Úvodný kvíz šikovnou automatikou navrhne doporučované stravovacie plány
- Bodové ohodnotenie jedál na základe „zdravosti“ a kompatibility so stravovacím plánom na ktorom práve sme
- Generovanie jedálničkov a možnosť zmeny jedál v ňom
- *Suggestic Lens*

Nevýhody

- Americká lokalizácia – mimo USA žiadne lepšie funkcie nefungujú
- Nie je možnosť si manuálne nastaviť kalórie a nutričné hodnoty!!!
- Väčšina stravovacích plánov je spoplatnená
- Aplikácia nijako neupravuje výber jedál podľa toho, aby sme dodržali nutričné hodnoty

2.1.2.5 Nutritionix

Nutritionix[20] je takisto americké riešenie ktoré si spomenieme ako posledné. Takisto ako spomínané ***Spoonacular*** a ***Edamam***, ich aplikácia nie je nič moc, a ani nemá veľa hviezdičiek v recenziách. Reálna sila sa skrýva v API. Veľa vecí je

[20] https://www.nutritionix.com/

podobných, ako napríklad NLP, preto spomeniem tie v čom sa líši od tých dvoch spomínaných. Výhodou tejto služby oproti dvom už spomínaným je integrácia **(A3)**, a to pomerne dosť solídna. Podľa ich web stránky (ku dňu 16.10.2020) sa jedná až o 171178 reštauračných položiek, čo je naozaj dych berúce. Nevýhodou oproti týmto dvom spomínaným je zasa absencia kategorizácie jedál až pri 99% produktov. Jediná rozumná kategorizácia ktorú ***Nutritionix*** implementuje tak je rozdelenie produktov na *Common* a *Branded*, avšak nič viac zložitejšie nemožno čakať. Takisto je problém s modulom **(A2)**, pretože žiadne recepty toto riešenie neponúka. Je síce možnosť vytvoriť si vlastné recepty, ale možnosť nájsť a vyhľadať nejaké iné recepty (napríklad hoci aj od iných používateľov) som tu nenašiel. Tak ako aj pri ostatných **(A4)** tu nenájdeme.

Výhody

- Asi najlepšia voľne dostupná databáza o produktoch a jedlách z reštaurácií na trhu (americkom)
- Rozdelenie produktov na *Common* a *Branded*

Nevýhody

- Žiadna kategorizácia podľa diéty či alergénov
- Integrácia **(A2)** je na žalostnej úrovni

2.1.2.6 Reštauračné aplikácie

V princípe akákoľvek známejšia aplikácia zameriavajúca sa na donášku, rozvoz či objednávky jedál cez internet, napr. **Bistro.sk**[21], **Dáme jídlo**[22], **Zomato**.

Tieto aplikácie nie sú síce priamym konkurentom na poli na ktorom sa snažíme niečo vytvoriť, avšak majú k tomu blízko tak sa hodí ich spomenúť spoločne s pár myšlienkami.

Keďže sa nejedná o nutričné manažéry, ale o aplikácie, zamerané na objednávky či rozvoz jedál, tak tu samozrejme nenájdeme modul **(A1)** a ani **(A2)**. Čo tu však nájdeme a vo veľmi dobrej forme, tak je modul **(A3)**. Obrovská výhoda týchto služieb oproti iným spomínaným aplikáciám sú ich dáta, týkajúce sa jedál v

[21] https://www.bistro.sk/

[22] https://www.damejidlo.cz

reštauráciách. Ostatné aplikácie zaostávajú v tom, že aj keď vedia o pozícii reštaurácie, pretože berú veľké databázy s polohami rôznych podnikov od tretích strán (ako napríklad už skôr spomínaný ***Foursquare***), tak už nevedia o ich ponúkaných jedlách a menu, resp. vedia to len vo veľmi obmedzenom množstve. Tieto aplikácie majú dáta o jedlách veľmi dobré a kvalitné. Preto ak by sme chceli aplikáciu využívať mimo USA, kde dáta na jedlá v reštauráciách sú pomerne dostatočné a zmapované, napríklad u nás doma, tak moja vízia je akási možnosť kolaborácie práve s týmito spoločnosťami, keďže oni už disponujú databázou ponúkaných jedál. Jediná nevýhoda takejto spolupráce môže byť v tom, že dané aplikácie zvyčajne nedisponujú dátami o nutričných hodnotách a kalóriách. To si však myslím, že v prípade, ak už dané spoločnosti spolupracujú s tými danými reštauráciami, ktoré majú vo svojej aplikácii, tak vyžiadať si nutričné hodnoty a kalórie k ponúkaným jedlám by až taký problém nebol, len to zatiaľ nebolo potrebné, tak sa to jednoducho neuskutočnilo.

Výhody

- Kdekoľvek kde chytáme internet, tak si vieme pozrieť reštaurácie v okolí a vybrať si niečo na čo máme chuť
- Možnosť objednávky a donášky domov
- Na základe recenzií sa vieme lepšie rozhodnúť
- Možnosť filtrovať podľa ceny, kvality, vzdialenosti...

Nevýhody

- Žiadne informácie o nutričných hodnotách jedál (jak online, tak ani priamo v reštauráciách - offline)
- Niekedy sa jedná o nezdigitalizované menu reštaurácií (je dostupná iba fotka menu) a preto nie je možné filtrovať/vyhľadávať podľa parametrov týkajúcich sa priamo pokrmu, ale skôr obecne parametrov týkajúcich sa reštaurácií či kategórie jedla na ktorú sa reštaurácia špecializuje

2.1.3 Zhrnutie

	A1	A2	A3	A4			
Kalorické tabuľky	✓	✓	X	X			
MyFitnessPal	✓	✓	+	X		✓	áno
Spoonacular/Edamam	✓	✓	X	X		+	čiastočne
Suggestic	✓	✓	✓	+		X	nie
Nutritionix	✓	X	✓	X			
Reštauračné aplikácie	X	X	✓	X			
NUTRIČNÝ ASISTENT	✓	+	+	✓			

Tak ako sme si stanovili v Úvode – Základné delenie, našou úlohou bude spraviť aplikáciu implementujúcu a spájajúcu dve hlavné časti **(A1)** a **(A4)**, spoločne s menej signifikantnými **(A2)** a **(A3)**, pretože aj keď existujú riešenia, ktoré spájajú prvé tri časti dohromady a niektorým sa to aj pekne darí, tak časť **(A4)** (ktorá bude to hlavné, čo odlíši našu prácu od ostatných) nie je v žiadnom produkte.

2.2 Dáta

Napriek tomu, že by naša aplikácia bola prelomová, ak by sme nemali tie správne dáta, tak by ju nikto nepoužíval. Je preto vhodné zamyslieť sa aj nad tým, čo máme k dispozícii a aké sú možnosti, keď dôjde reč na dáta.

Zodpovedzme si najprv otázku, na čo všetko dáta (v našej aplikácii) vlastne potrebujeme? Minimálny nutný základ sú dáta s receptami a dáta o potravinách a polotovaroch a k nim prináležiace nutričné hodnoty a kalórie. Ďalšia vec ktorá by sa nám hodila a na ktorej môžeme postupne vylepšovať našu aplikáciu tak sú dáta z reštaurácií, konkrétne teda ponúkané jedlá a k nim tiež prináležiace nutričné hodnoty spolu s kalóriami. Ak by sme toto všetko mali, tak z 80% už máme vyhraté. Zvyšných 20% by nám tvorili veci ako prináležiace čiarové kódy k potravinám a polotovarom, kategorizácia jedál a receptov, prípadné dáta o mikroživinách (vitamíny, minerály...) a iné. Nie je to niečo, bez čoho by sme sa v našej práci nezaobišli, ale všetko čo máme naviac nás určite poteší, nakoľko s tým vieme lepšie pracovať a eventuálne sa snažiť o lepšiu mieru personalizácie do budúcnosti.

2.2.1 Kritériá hodnotenia

Tak ako v prípade analýzy konkurencie, aj v tomto prípade si potrebujeme určiť nejaké dôležité kritériá na základe ktorých budeme poskytované dáta hodnotiť

a rozhodovať sa. Najdôležitejším kritériom vôbec je pre nás voľná dostupnosť dát. To znamená, že dáta musia byť buď voľne stiahnuteľné, alebo prístupné verejným API. Ak táto podmienka nie je splnená, tak dáta môžu byť akokoľvek kvalitné, pre nás sú ale tým pádom nepoužiteľné v našej práci. Ďalší faktor, ktorý potrebujeme zohľadniť v našej práci, tak je potreba dát jedál všetkých troch druhov – produkty, recepty a aj reštauračné jedlá. Recepty potrebujeme kvôli generovaniu jedálničkov, produkty potrebujeme kvôli tomu, aby naša aplikácia mohla fungovať aj ako bežný nutričný manažér, a reštauračné jedlá potrebujeme kvôli tomu, aby sme ukázali zaujímavú funkcionalitu aplikácie, keď takéto dáta máme k dispozícii. Samozrejme, k týmto jedlám neodmysliteľne patrí nutnosť informácie o nutričných hodnotách. V našej práci mikronutrienty brať v úvahu nebudeme, takže nám stačí, ak dáta obsahujú informácie o makronutrientoch (kalórie, tuky, sacharidy, bielkoviny). Posledný dôležitý faktor je veľkosť databázy. Ak by naše používané dáta neboli dostatočne veľké, aplikáciu by sme nedokázali plnohodnotne demonštrovať. Spomínané faktory sú teda tými hlavnými, na základe ktorých sa budeme rozhodovať pre konečnú voľbu datasetu.

2.2.2 Dostupné dáta

Najprv si predstavíme lokálne dáta. Okrem tých lokálnych riešení sa však zameriame aj na zahraničné riešenia. Spomínané zahraničné riešenia budú lokalizované najmä pre americký trh, pretože tam je výber omnoho lepší a dostupnosť dát väčšia. Pre reálne použitie aplikácie v našich podmienkach nebudú mať americké dáta veľkú reálnu hodnotu (najmä reštauračné), no pre demonštráciu schopností novej a inovatívnej funkcionality aplikácie je možné ich použiť.

2.2.2.1 Nutridatabaze.cz

Nutridatabaze.cz[23] je česká databáza potravín pod taktovkou Ministerstva zemědelství ČR. Je voľne k dispozícii a stiahnuteľná. Potraviny obsahujú všetky potrebné údaje, teda makroživiny a kalórie. Čo tu nenájdeme sú mikroživiny, no hlavným limitujúcim faktorom je veľkosť databázy, a to <1000 produktov. Zároveň sa databáza zameriava iba na samotné produkty resp. potraviny a preto tu nenájdeme

[23] https://www.nutridatabaze.cz/ - Databáze složení potravin České republiky

žiaden popis receptov. Pre skúšobné účely je však postačujúca a preto bola táto databáza využitá pri vývoji aplikácie.

2.2.2.2 Kalorické tabuľky

Kalorické tabuľky keďže sú hotový produkt, tak tiež majú svoju databázu dát o potravinách a ich nutričných hodnotách. V databáze nájdeme takisto aj recepty jedál, čo je určite plus. Databáza je pomerne veľká; ku dňu 23.6.2020 mali v databáze zaznamenaných 213808 potravín. Keďže je to lokalizovaná aplikácia, tak sa zdá, že sa jedná o horúceho kandidáta na to, aby sme dáta použili. Problémom je však to, že nemajú verejné API a tým pádom nejde nijak rozumne jednoducho pracovať s ich databázou.

2.2.2.3 FoodData Central

FoodData Central[24] je databáza ministerstva poľnohospodárstva Spojených štátov amerických. Jedná sa v podstate o podobnú databázu ako spomínaný český ekvivalent, ale s tým, že táto databáza je omnoho väčšia. Ku dňu 23.6.2020 obsahuje 334058 záznamov. Takisto sa jedná o databázu surovín, polotovarov a produktov, takže v nej nenájdeme žiadne recepty a reštauračné jedlá. Na rozdiel od ***Nutridatabaze.cz***, táto databáza obsahuje aj mikroživiny, takže tu nájdeme aj obsah vitamínov a minerálov obsiahnutý v potravinách. Databáza je verejná, voľne dostupná a dokonca je k dispozícii aj verejné API.

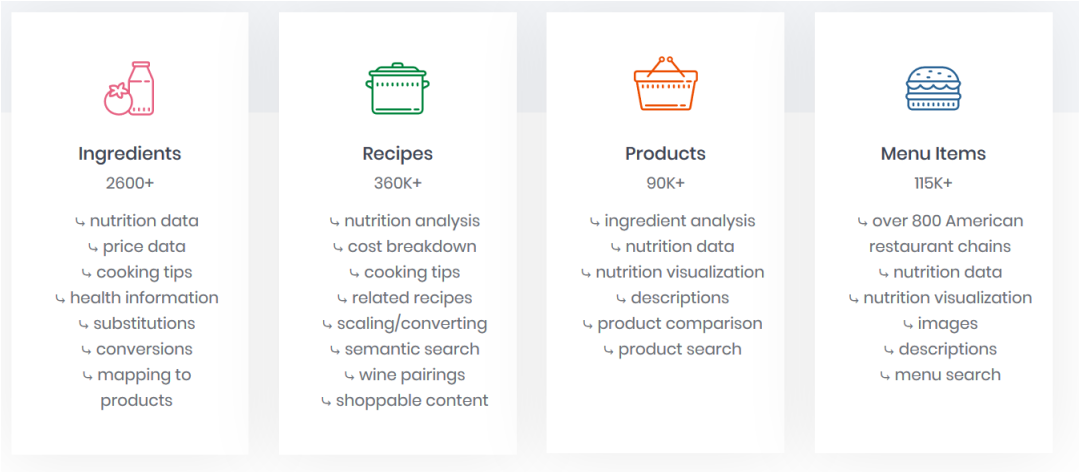
2.2.2.4 Spoonacular, Edamam, Suggestic, Nutritionix

Tieto súkromné americké riešenia sme si už spomínali vyššie v analýze konkurencie. Spoločne so svojimi vlastnými aplikáciami ponúkajú aj svoje databázy. Preberieme si ich spoločne, pretože sú si vo veľa veciach podobné a pri osobitnom rozoberaní by sme sa zbytočne viackrát iba opakovali. Poukážeme radšej na ich rozdiely a v čom má ktorá služba od ktorej navrch, a zároveň sa zameriame na to, čo je primárne dôležité pre nás.

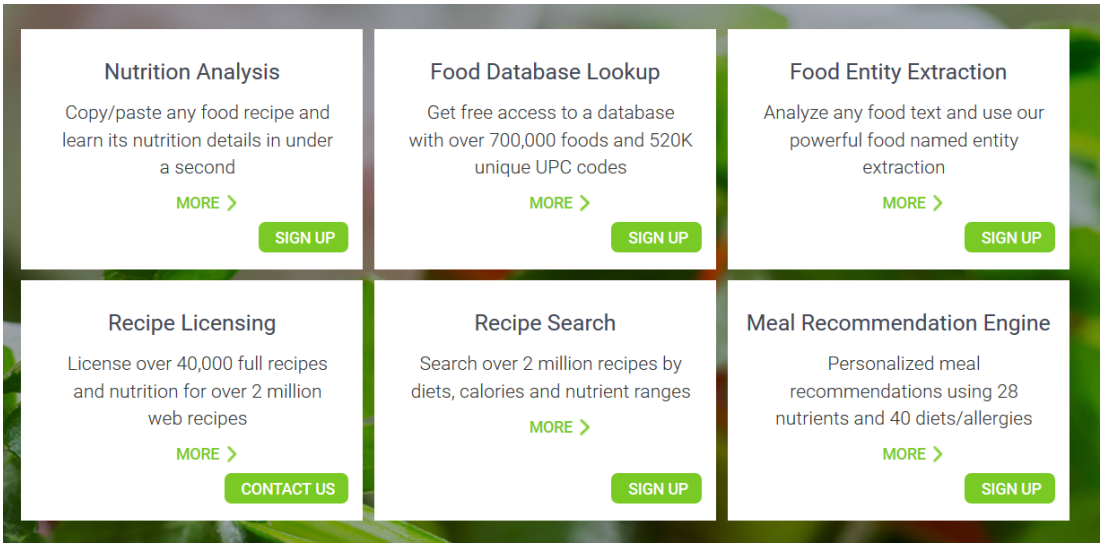
Vo svojej podstate sú si podobné; všetky obsahujú databázu potravín a ich nutričných hodnôt a okrem ***Nutritionix*** obsahujú aj databázu receptov. ***Suggestic*** a

[24] https://fdc.nal.usda.gov/fdc-app.html - Databáza potravín - U.S. Department of Agriculture

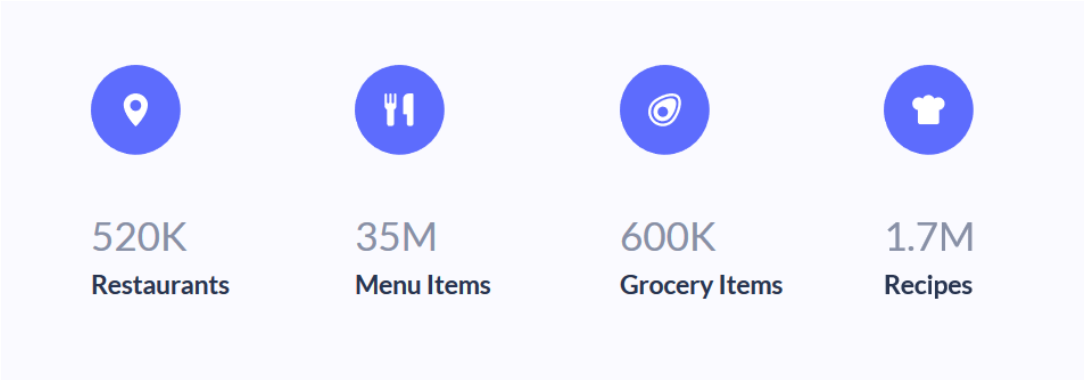
Nutritionix majú oproti prvým dvom menovaným k dispozícii aj databázu reštaurácií spoločne s jedlami a k nim prináležiacimi nutričnými hodnotami. Toto je skutočnosť, ktorá momentálne službe ***Suggestic*** nahráva do karát v našom následnom využití, nakoľko do našej aplikácie chceme zakomponovať okrem potravín a jedál z receptov aj dáta z reštaurácií a podľa zatiaľ spomínaných informácií, ***Suggestic*** disponuje všetkými tromi. Pre predstavu si uvedieme aj jednotlivé hodnoty veľkosti databáz, aké uvádzajú konkrétne riešenia:



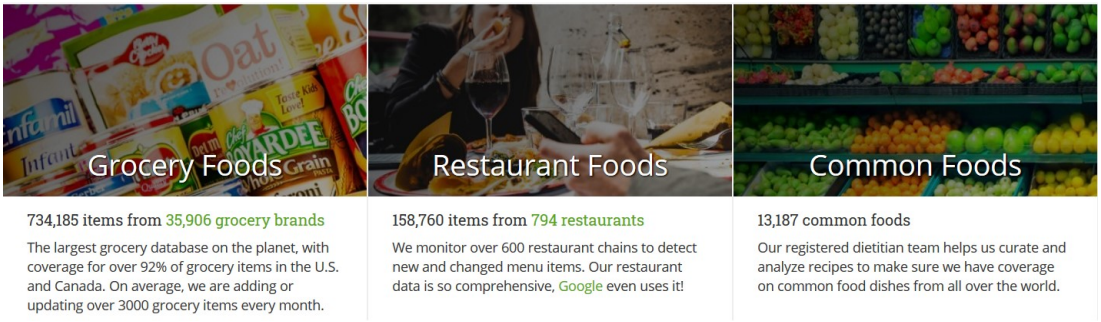
Obrázok 3: Spoonacular - https://spoonacular.com/food-api



Obrázok 4: Edamam - https://developer.edamam.com/



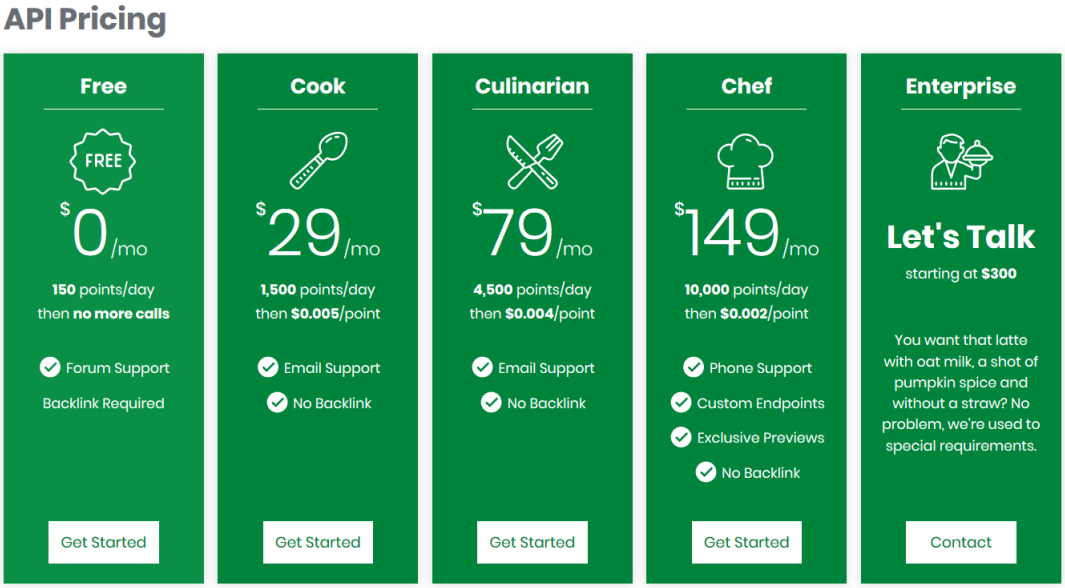
Obrázok 5: Suggestic - https://www.suggestic.com/



Obrázok 6: Nutritionix - https://www.nutritionix.com/database

Z priložených obrázkov vidíme, že každé jedno riešenie nám ponúka viac než potrebné množstvo dát, ktoré je pre nás v rámci tejto práce bohato dostatočné.

Na čo však nemôžeme zabudnúť tak je náš kľúčový faktor v konečnom rozhodovaní a výbere toho ktorého riešenia a jeho databázy pre našu prácu, a to aké API jednotlivé riešenia ponúkajú a aké podmienky sa k ich použitiu viažu. Z doterajšej analýzy vyzerá, že ***Suggestic*** má pravdepodobne najprepracovanejšie dáta; zásadný problém je však v tom, že ***Suggestic*** neponúka žiadne voľne dostupné API. Možno by sa s nimi dalo na niečom dohodnúť, to však nie je vôbec nutné a ani potrebné, nakoľko existujú alternatívy. Zvyšné tri riešenia ponúkajú okrem spoplatnených služieb aj možnosť bezplatnej verzie API ktorá funguje v obmedzenom režime. Pre demonštračné účely nám bude stačiť. Povedzme si, čo to konkrétne znamená:

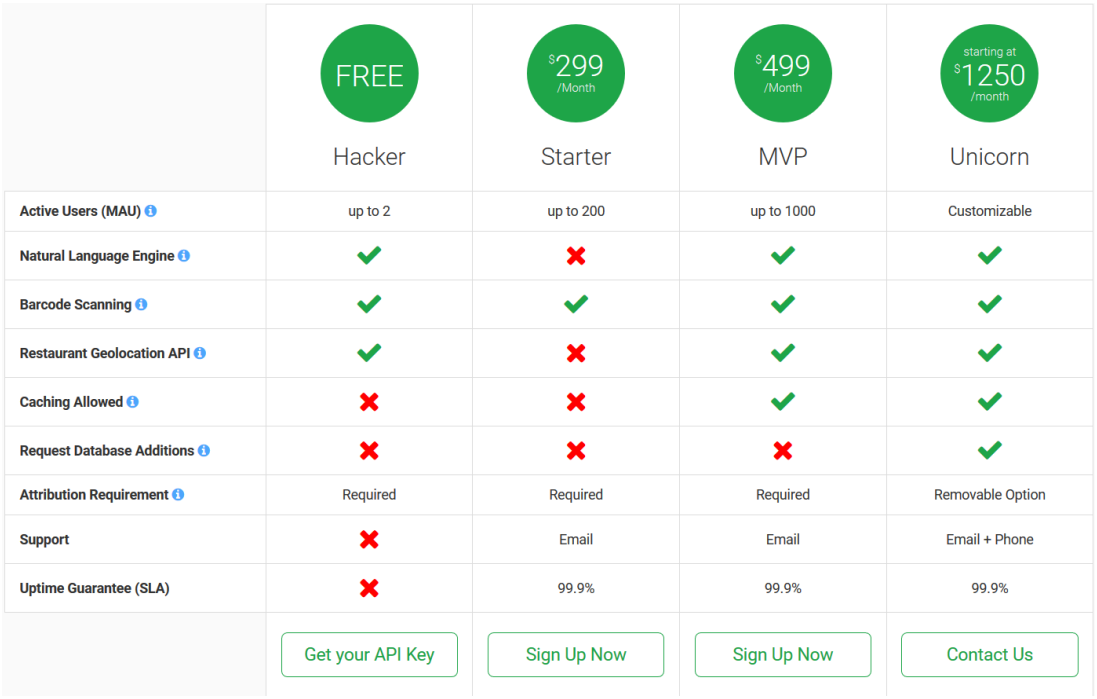


Obrázok 7: Spoonacular pricing - https://spoonacular.com/food-api/pricing

Spoonacular ponúka k dispozícii plnú verziu svojho API. Nie je teda v tomto smere nijak obmedzená a môžeme naplno využívať všetku jej funkcionalitu. Obmedzenie v tomto prípade je maximálny počet 150 volaní na deň. To samozrejme nie je veľa, ale keď to nejak rozumne skombinujeme s iným API, tak by nám to mohlo vystačiť. ***Spoonacular*** API obsahuje všetko čo potrebujeme a dokonca si dovolím tvrdiť, že z týchto troch spomínaných (***Suggestic*** už nerátame v úvahu) je najkvalitnejšie. Obsahuje veľa informácií o jedlách a produktoch. Makro a mikronutrienty sú samozrejmosťou, kategorizáciu majú veľmi rozsiahlu, napríklad v rámci alergénov, alebo pri receptoch napríklad majú kategorizáciu jedál do chodov (raňajky, hlavný chod, snack...). Toto API teda vyzerá ako solídny kandidát.

Edamam má na každú službu (Obrázok 2) osobitné API. Vyhľadávanie receptov alebo potravín (aj keď s využitím rozdielneho API) podlieha podobným podmienkam ako FREE verzia ***Spoonacular***. Obsahuje obmedzenie na počet volaní za mesiac a takisto obmedzenie na počet volaní za minútu. Pri potravinách je to 10 za minútu, pri receptoch je to 5 za minútu a dokopy 5000 za mesiac.

Nutritionix na svojich stránkach neposkytuje informácie o obmedzení počtu API volaní na deň. Po emailovej komunikácii nám však bola poskytnutá informácia, že limit je 200 volaní na deň. Okrem toho je tam aj obmedzenie na maximálne aktívne používanie dvoma používateľmi naraz, avšak to pri vývoji a ani na demonštračné účely vôbec nevadí.



	FREE Hacker	$299 /Month Starter	$499 /Month MVP	starting at $1250 /month Unicorn
Active Users (MAU)	up to 2	up to 200	up to 1000	Customizable
Natural Language Engine	✓	✗	✓	✓
Barcode Scanning	✓	✓	✓	✓
Restaurant Geolocation API	✓	✗	✓	✓
Caching Allowed	✗	✗	✓	✓
Request Database Additions	✗	✗	✗	✓
Attribution Requirement	Required	Required	Required	Removable Option
Support	✗	Email	Email	Email + Phone
Uptime Guarantee (SLA)	✗	99.9%	99.9%	99.9%
	Get your API Key	Sign Up Now	Sign Up Now	Contact Us

Obrázok 8: Nutritionix pricing - https://www.nutritionix.com/business/api

Dôležitá vec, ktorú treba pri ***Nutritionix*** spomenúť, je disponovanie dátami o reštauráciách a ponúkaných jedlách. Tie dáta síce nie sú dokonalé, nakoľko ich API funguje trochu divným spôsobom a nevieme sa dostať ku všetkým ponúkaným jedlám, ale vždy iba k 20 najčastejším pre danú reštauráciu, no stále je to niečo, s čím môžeme pracovať. Z teoretického pohľadu je nám vlastne jedno, že aké jedlá budú medzi tými 20 najčastejšími. Dôležitý je fakt, že máme dáta o polohách reštaurácií a nimi ponúkané jedlá. Tým pádom môžeme tieto dáta integrovať do našej aplikácie a demonštrovať funkčnosť a použiteľnosť aplikácie pri disponovaní dátami z reštaurácií.

Hlavný problém ***Nutritionix*** je však absencia receptov a kategórií. Tým pádom nemôžeme siahnuť iba po tomto riešení a spoliehať sa len na neho, pretože na generovanie jedálničkov v module **(A4)** potrebujeme recepty spoločne s kategorizáciou, či sa jedná o recept na raňajky, na obed, na večeru...

2.2.2.5 Zomato

Zomato tiež ponúka databázu k dispozícii, v tomto prípade sa však jedná o trochu iné dáta, než sme doposiaľ rozoberali. Keďže ***Zomato*** je aplikácia, ktorá rieši vyhľadávanie reštaurácií v okolí, prehľad ponuky ich jedál a prípadné recenzovanie, tak dáta ktoré ***Zomato*** ponúka sa týkajú tejto problematiky. Ponúkajú voľne dostupné

API, ktoré je obmedzené na 1000 volaní za deň. Podľa ich stránky s pomocou tohto API by sme mali prístup k vyše 1,5 milióna reštauráciám roztrúsených po 10 000 mestách po celom svete (a aj v našich lokalitách, čo je veľká výhoda)[25]. Keďže sa chceme zamerať aj na navrhovanie jedál s ohľadom na pozíciu užívateľa a možnosť stravovať sa v reštauráciách, nie je zlý nápad zvážiť kombináciu tohto API s iným API zameraným na potraviny a recepty. Problém v tomto prípade však nastáva pri nutričných hodnotách. ***Zomato*** totiž nevedie nutričné hodnoty a kalórie pri jedlách. V niektorých prípadoch dokonca nenájdeme ani zdigitalizované menu reštaurácií.

2.2.2.6 Open food facts

Open food facts[26] je databáza ktorú by bolo dobré na záver porovnania rôznych dátových možností spomenúť. ***Open food facts*** je open-source globálna databáza potravín, ktorá funguje na jednoduchom crowdsourcingovom[27] princípe. Rovnako ako v prípade americkej alebo českej národnej databázy, ani v tomto prípade tu nenájdeme recepty, to však neznamená, že táto databáza nemá potenciál. Ku dňu 24.6.202 táto databáza obsahuje 1,398,194 produktov, čo je viac než akákoľvek iná vyššie spomínaná databáza. Dôležitý potenciál tejto databázy vidím v jej možnosti použitia v prípade globalizácie aplikácie. Otvorené a voľne dostupné API je samozrejmosťou, rovnako ako aj možnosť priameho stiahnutia databázy v rozličných formátoch.

[25] https://developers.zomato.com/api

[26] https://world.openfoodfacts.org/ - opensource globálna databáza produktov

[27] https://cs.wikipedia.org/wiki/Crowdsourcing

2.2.3 Zhrnutie

	Nutritionix	Spoonacular	Edamam	FoodData Central	Suggestic	Open Food Facts	Nutridatabaze.cz	Zomato	Kalorické tabuľky
Free API	✓	✓	✓	✓	X	✓	✓	✓	X
Macronutrients	✓	✓	✓	✓	✓	✓	✓	X	✓
Micronutrients	✓	✓	✓	✓	✓	X	✓	X	✓
Allergens	X	✓	✓	✓	✓	✓	X	✓	X
Categories	X	✓	✓	X	✓	✓	X	✓	✓
Branded/Common	✓	X	X	✓	✓	✓	X	X	✓
Restaurants	✓	X	X	X	✓	X	X	✓	X
Premade recipes	X	✓	✓	X	✓	X	X	X	✓

Tabuľka 1: Zhrnutie porovnávaných dátových služieb podľa kategórií

3. Špecifikácia

Naša aplikácia bude pozostávať zo 4 modulov. V Úvode – Základné delenie sme už načrtli čo má ktorý modul robiť a na čo bude slúžiť; v tejto kapitole si každý jeden modul rozoberieme osobitne, povieme si aké na neho máme požiadavky, aké funkcie modul obsahovať musí (must-have), aké funkcie by bolo fajn ak by obsahoval (nice-to-have) a aké funkcie obsahovať nebude. Pri moduloch **(A2)** a **(A3)**, napriek tomu že v našej práci nebudú vyslovene osobitne implementované, tak si popíšeme, čo by mal dobrý a plnohodnotný modul **(A2)** a **(A3)** implementovať v porovnaní ku konkurencii. Nakoľko sme ich brali ako kritériá pri porovnávaní s konkurenciou, tak ich považujeme za základné moduly, ktorých dokončenie by bol prvý krok, ak by sme sa rozhodli našu prácu rozširovať. Podrobnej diskusii použitia konkrétnych technológií a spôsobu riešenia problému na základe špecifikácie sa budeme venovať v nasledujúcej kapitole Dizajn.

3.1 Kľúčové body

V tejto práci našu pozornosť nebudeme smerovať na funkcie, ktoré už vymyslela konkurencia (ak pre našu aplikáciu nie sú vitálne), na UI a na UX. Užívateľské prostredie našej aplikácie bude iba proof-of-concept na demonštráciu modulu **(A4)** a nebudeme sa snažiť o žiadne nadmerné grafické spracovanie či farebné zladenie. V našom prípade sa zameriame len na vývoj mobilnej aplikácie, keďže v dnešnej dobe je to už 50.81% celkového prístupu k internetu, čo sa deje prostredníctvom mobilných zariadení[28]. O čo sa naopak budeme snažiť, tak to je modulárnosť aplikácie, dobrá integrovateľnosť jednotlivých modulov do seba, dbať na možnosť rozšírenia existujúcich modulov v budúcnosti, používanie

[28] https://www.statista.com/statistics/277125/share-of-website-traffic-coming-from-mobile-devices/#:~:text=Mobile%20accounts%20for%20approximately%20half,since%20the%20beginning%20of%202017

programátorskej filozofie „divide et impera[29]“ kdekoľvek, kde to bude možné a samozrejme naprogramovanie modulu **(A4)**.

3.2 Modul A1 – Nutričný manager

Modul A1 spoločne v súčinnosti s modulom A4 tvoria základ našej aplikácie. Modul A1 slúži ako hlavný ovládací prvok, ktorý obsluhuje chod aplikácie a zabezpečuje správnu komunikáciu front-endu s back-endom aplikácie. To v praxi znamená, že v prípade ak si interakcia užívateľa s aplikáciou vyžaduje použitie funkcie akéhokoľvek iného modulu, tak modul A1 zabezpečí sprostredkovanie tejto funkcionality a prípadné zobrazenie výsledku na front-end. Je to teda práve modul A1, prostredníctvom ktorého užívateľ môže jednoducho využívať ostatné moduly.

Tento modul je esenciálnou verziou konkurenčných produktov. To znamená, že obsahuje všetko čo je nutné a potrebné pre zapisovanie a manažovanie prijatých potravín a ich nutričných hodnôt spolu s kalóriami. Medzi funkcie, ktoré tento modul implementuje (must-have), patrí:

- **A.** Zobrazovanie prijatých kalórií a nutričných hodnôt (vrátane prezerania zapísaných jedál)
- **B.** Zapisovanie prijatej potravy/zjedených jedál
- **C.** Nastavenie limitov pre nutričné hodnoty a kalórie (automaticky alebo manuálne)
- **D.** Nastavenie údajov zadaných užívateľom (váha, výška, vek, aktivita...)
- **E.** Spravovanie užívateľského prostredia
- **F.** Prezeranie databázy jedál s možnosťou filtrovania

C, D – aj keď automatické nastavenie kalórií nie je nutné pre preukázanie funkčnosti našej práce, bude lepšie a intuitívnejšie, ak aplikácia bude zahŕňať aj automatický výpočet nutričných hodnôt a kalórií na základe užívateľom zadaných parametrov. To zapadá do filozofie tejto aplikácie, že nemusíme o jedle a fitness vedieť nič; stačí

[29] https://en.wikipedia.org/wiki/Divide_and_rule - rozdeľuj a panuj. Väčšie problémy si rozdelíme na menšie a budeme sa ich snažiť riešiť. V praxi pre našu aplikáciu to znamená, že na každý problém sa budeme snažiť vytvoriť vlastný modul, aby aplikácia bola čo najmodulárnejšia.

nám odhodlanie. Medzi parametre ktoré aplikácia používa na výpočet v aktuálnej podobe zaradzujeme váhu, výšku, vek, pohlavie, aktivitu počas dňa + pravidelnosť cvičenia a to či chceme chudnúť, priberať alebo udržať aktuálnu hmotnosť.

E – tento modul slúži ako sprostredkovateľ užívateľského prostredia. Väčšinu práce ktorú bude užívateľ vykonávať, tak ju bude robiť práve tu. To v praxi znamená, že tento modul implementuje väčšinu front-endu ktorý naša aplikácia bude používať; ostatné moduly budú naprogramované tak, aby sme ich napasovali do tohto modulu a do jeho front-endu. Pre príklad, pri plnej implementácii by mohol byť front-end modulu **(A2)** nejaká jednoduchá mriežka, pri **(A3)** zase mapa.

Ďalšie funkcie ktoré by mohol modul **(A1)** v budúcnosti implementovať (nice-to-have) sú nasledovné:

- Priama integrácia natural language processing do našej aplikácie (keďže už existuje v konkurencii). Hlavný problém ktorý by sa tu musel vyriešiť by však bola viacjazyčnosť, keďže NLP stojí na tom, ako dobre dokážeme ktorý jazyk počítačovo spracovávať, a je jasné, že angličtina má v tomto navrch oproti všetkým ostatným jazykom. Bolo by to však určite nice-to-have mať takúto funkciu a preto sme ju tu dali.
- Image recognition resp. rozpoznávanie obrazu. Použitie takejto funkcie by dokázalo extrémne zjednodušiť akékoľvek monitorovanie prijatej potravy a k nim korešpondujúcich nutričných hodnôt. Ak by sme dokázali spraviť iba fotografiu jedla a naša aplikácia by nám rovno zapísala dané jedlo, bolo by to omnoho rýchlejšie, jednoduchšie a aj pre užívateľov náramne prívetivejšie, než spôsob, akým je to dnes robené všade. Dnes si užívateľ musí bohužiaľ dať aspoň tú námahu otvoriť si nejaký searchbar, vyhľadať v ňom ním skonzumované jedlo, kliknúť na neho a dané jedlo si zapísať do aplikácie, ktorú používa. Samotné rozpoznávanie obrazu už v dnešnej dobe existuje (Caloriemama[30], IBM Watson[31]) avšak čo je problém v našom prípade, tak je to to, že nám nestačí iba rozpoznať obraz, no pre nás je dôležité vedieť aj odhadnúť váhu toho jedla. Caloriemama vyzerá, že tento problém nejako rieši, avšak ich API je platené, preto sme ho ani my v našej práci nevyužívali.

[30] https://www.caloriemama.ai/

[31] https://github.com/IBM/watson-calorie-counter

Ak by existovalo rozpoznávanie obrazu jedla s rozpoznávaním jeho váhy, bol by to veľký pokrok pre toto odvetvie a sektor IT. Preto sme aj tento nápad zaradili do sekcie nice-to-have, pretože je to niečo, čo by sa naozaj oplatilo mať v aplikácii.

3.3 Modul A2 – Receptár

Modul A2 je sekundárny modul, ktorý má na starosti časť s receptami. Od tohto modulu v našej práci požadujeme to, aby dokázal spracovávať a filtrovať recepty z databázy. Modul A2 by mal byť schopný samostatného fungovania, nakoľko práca s receptami je výlučne jeho záležitosť a takto môže implementovať rôznu funkcionalitu nad používanými receptami. V našej aplikácii však z dôvodu jednoduchosti modul A2 nebude implementovaný ako samostatný modul, no v kapitole Architektúra si povieme, ako sme navrhli našu aplikáciu tak, aby sa zohľadňovalo plnohodnotné doimplementovanie modulu A2 v budúcnosti.

V našej práci z modulu A2 budeme implementovať:

A. Jednoduché filtrovanie, jednoduché prehliadanie receptov

A – užívateľ bude mať možnosť prehliadať si recepty a zároveň bude mať k dispozícii jednoduchú filtráciu podľa nutričných hodnôt. Pre účely toho, čo sa našou prácou snažíme dokázať, a to je demonštrácia **(A4)**, viac nepotrebujeme. Stačí, že máme k dispozícii recepty, ktoré budeme využívať na generovanie jedálničkov v module **(A4)** alebo na vlastný zápis/pridanie do nutričného manažéra **(A1)**.

Čo v našej práci síce implementovať nebudeme, ale boli by to funkcie, ktoré by sme ako prvé doimplementovali pri rozširovaní modulu:

B. Kategorizácia jedál

C. Možnosť hodnotiť prípadne recenzovať jednotlivé recepty s vizualizáciou počtu hodnotení

B – čo určite od tohto modulu chceme v budúcnosti vidieť, tak je možnosť vedieť si recepty nejakým spôsobom filtrovať podľa rozmanitejších kategórií, ako sú alergény, rôzne diéty a podobne. V našej práci budeme mať iba filtráciu podľa nutričných hodnôt nakoľko možnosti filtrácie receptov pri manuálnom zadávaní do modulu **(A1)** nijako neovplyvňujú algoritmy bežiace v module **(A4)**. Bola by to však jedna z

prvých vecí, ktoré by sme dodali do tejto aplikácie, keby sme ju chceli vydať ako finálny produkt.

C – táto vlastnosť je veľmi prospešná pre úspešné fungovanie aplikácie aj mimo akademickej pôdy. Preto tu na to ostane priestor, no v našej práci sa touto funkciou zaoberať nebudeme

Nakoniec, čo by bolo nice-to-have v našej práci:

D. Zaujímavým nápadom pri kategorizácii by bolo rozdelenie receptov na zdravé a nezdravé. Nie vždy nám ide iba o zdravé stravovanie, a niekedy keď nemáme nič iné k dispozícii, tak sme ochotný zjesť aj niečo nezdravé, pri čom nám však ostanú kalorické a nutričné limity v norme.

E. Mať možnosť pridávať vlastné recepty

F. Zaujímavým doplnením by mohlo byť rozdelenie receptov na *Verified* a *Community*. *Verified* by mohli byť akési recepty od nás od výrobcu aplikácie, ktoré sú overené, a *Community* by boli recepty z komunity používateľov, teda tie, ktoré by si tam mohli vkladať samotní používatelia.

3.4 Modul A3 – Reštaurácie

Modul A3 je druhý sekundárny modul našej aplikácie. Tento modul má na starosti sprostredkovanie dát ohľadom reštaurácií a ich jedál či menu. Jeho funkčnosť by mohla byť v plnej implementácii podobná akejkoľvek inej implementácii **(A3)** v konkurencii, napríklad **Nutritionix – Restaurant Nutrition Map (beta)**[32]. Rovnako ako aj modul A2, ani tento modul nebudeme v našej práci implementovať (kvôli jednoduchosti) ako samostatný osobitný modul.

V našej práci budeme implementovať:

A. Vyhľadávanie reštauračných jedál z okolia na základe zvoleného rádiusu a jednoduchá filtrácia pomocou nutričných hodnôt

[32] https://www.nutritionix.com/restaurant-map

Čo v našej práci síce implementovať nebudeme, ale boli by to funkcie, ktoré by sme ako prvé doimplementovali pri rozširovaní modulu, rovnako, ako pri module **(A2)**:

B. Ukážka zvoleného jedla na mape

C. Možnosť vyhľadávať reštaurácie na mape spoločne s nimi ponúkanými jedlami a ich nutričnými hodnotami + integrácia s GPS

C – užívateľ by mal k dispozícii mapu na ktorej bude môcť vyhľadávať reštaurácie spoločne s nimi ponúkanými jedlami a ich nutričnými hodnotami. Mal by mať možnosť filtrovať podľa aktuálnej pozície (GPS) a zvoleného rádiusu na mape.

3.5 Modul A4 – Guidance bot

Modul A4 bude tým, čo nás a našu aplikáciu odlíši od existujúcich produktov na trhu (ako sme už vydedukovali v sekcii Zhrnutie analýzy konkurencie). V tejto sekcii si zašpecifikujeme presne aké sú naše požiadavky na tento modul a vytýčime si konkrétne body čo má tento modul robiť, aby dosahoval také výsledky, aké v našej aplikácii chceme:

A. Generovanie jedálničku

B. Ad-hoc prístup pri zadaní nenavrhovaného jedla

C. Integrácia stravovania v reštauráciách do jedálnička

D. Výpočet matematického modelu jedálnička na základe vstupných parametrov užívateľa

E. Komunikácia s databázou

A – *Guidance bot* bude schopný vygenerovať jedálniček na celý deň, a to tak, aby boli zachované potrebné nutričné limity pre užívateľa. V rámci tejto funkcionality takisto požadujeme, aby bol *bot* schopný zmeniť (pregenerovať) istú menšiu časť dňa, a to napríklad zmeniť iba obed a večeru, no zbytok nechať nedotknutý. Toto požadujeme kvôli personalizácii jedálnička - každému môže vyhovovať iné jedlo. Jedálniček bude rozdelený na 4 jedlá, a to raňajky, obed, večeru a snack. Principiálne nie je žiadny problém v tom nastaviť tých jedál viac alebo menej, no kvôli

zobrazovaniu vo front-ende sme si museli zvoliť nejakú konštantu. My budeme v našej práci pracovať so 4 chodmi do dňa.

B – keďže chceme stavať na tom, že užívateľ nemusí striktne dodržiavať nami vygenerovaný jedálniček, tak je potrebné, aby *bot* dokázal zhodnotiť situáciu ad-hoc a podľa toho nastaviť jedálniček. Táto situácia nastane vždy vtedy, keď užívateľ zadá do aplikácie nenavrhované jedlo. Keďže nenavrhované jedlo (v jedálničku) je niečo, s čím sa za optimálnych podmienok nepočíta, je potrebné sa nejak s týmto momentom vysporiadať, aby nutričné limity ostali stále zachované. Pre toto ad-hoc rozhodovanie v našom projekte použijeme 3 možnosti, ktoré budú užívateľovi dostupné po zadaní nenavrhovaného jedla do denného príjmu:

1. *Cheatday* – prvá funkcia, ktorou náš *bot* bude disponovať je funkcia ignorancie. Ak si užívateľ nepraje, aby ho *bot* korigoval, musíme to rešpektovať. Jedna z možností teda bude, aby *bot* nerobil nič a žiadne úpravy alebo zmeny do konca dňa už vykonávať nebude. Užívateľ si môže zapisovať aké jedlá chce a koľko chce a *bot* na to nebude nijak ad-hoc reagovať.
2. *Nextday* – v tomto prípade bude funkcionalita na prvý pohľad podobná prvej možnosti, no tu sa udeje to, že akýkoľvek eventuálny prebytok nutričných hodnôt (ktorý by mohol vzniknúť práve tým, že si do nášho zoznamu zjedených jedál vložíme nenavrhované jedlo) sa odráta od zajtrajšieho jedálnička, ktorý *bot* vygeneruje. V praxi to teda znamená, že ak by sme v daný deň prijali napríklad 2300kcal, pričom náš denný príjem kalórií by nemal prekročiť hranicu 2000kcal, tak tento nadbytok 300kcal reflektujeme do vygenerovania jedálnička na ďalší deň. Tým pádom na ďalší deň nám *bot* vygeneruje jedálniček, ktorý bude mať dokopy iba 1700kcal a vytvorí nám kalorický deficit, ktorý sa zneguje s kalorickým nadbytkom z minulého dňa.
3. *Thisday* – tretia funkcia ktorou náš *bot* disponuje je možnosť pregenerovať jedálniček ad-hoc ešte na ten istý deň, tak, že po zjedení nových vygenerovaných jedál ešte stále dodržíme stanovené limity nutričných hodnôt. Vygenerované jedlo pre chod, do ktorého sme si vložili jedlo bude ignorované a pregenerujú sa nám iba chody, kde nie je

žiadne zadané jedlo. Rovnako, tie vygenerované jedlá ktoré sme už zaškrtli a označili ako zjedené, pregenerované nebudú.

C – pre veľa ľudí je stravovanie v reštauráciách neodmysliteľná súčasť ich (pracovného) života. Preto, ak chceme kvalitnú aplikáciu na dodržiavanie nutričných limitov, tak bez integrácie stravovania v reštauráciách sa nezaobídeme. Kvôli tomu chceme dať užívateľovi možnosť vložiť do svojho jedálnička aj jedlá z reštaurácií. V našom prípade bude možnosť zmeny jedálničku tak, aby sme doňho inkorporovali stravovanie vonku v reštaurácii (na to však nebude vyslovene osobitná funkcia, pretože to vieme simulovať pomocou výberu jedla z reštaurácie v okolí, zápise jedla do jedálnička a potom na základe zvolenia možnosti *Thisday* budeme mať vytvorený nový jedálniček, ktorý inkorporuje to dané zvolené jedlo do nášho jedálnička).

D – aby sme vôbec dokázali podávať správne a uspokojivé výsledky, je potrebné spraviť dobrý matematický model jedálnička. Užívateľ zadá 6 parametrov (v budúcnosti ich môže byť viac), a to pohlavie, vek, výška, váha, aktivita životného štýlu a cieľ (pribrať, schudnúť, udržať hmotnosť) a na ich základe vytvoríme limity v rámci nutričných hodnôt a kalórií pre celý deň, ako aj pre jednotlivé chody na mieru, pre užívateľa. Viac o tom, ako presne funguje výpočet modelu sa dozvieme v Implementácii – Matematický model.

E – napriek tomu, že užívateľovi je jedno, akým spôsobom sa k nemu dáta dostanú, tak pre nás ako programátorov je rýchlosť a spôsob akým sa dostanú dáta od zdroja k cieľu častokrát rozhodujúcim faktorom, či sa užívateľ po prvej minúte používania aplikácie rozhodne aplikáciu odinštalovať, alebo si ju ponechá a dá jej šancu. *Guidance bot* má preto okrem výpočtov na generovanie jedálnička a matematického modelu na starosti ešte aj komunikáciu s databázou, z ktorej sa berú dáta do našej aplikácie. Prečo padlo takéto rozhodnutie bude popísané v časti Architektúra modulu A4.

Povedzme si zopár nápadov, ktoré sa síce momentálne do rozsahu tejto práce nezmestia, no v budúcnosti by sa nimi rozhodne dal tento modul A4 vylepšiť:

- Integrácia kalendára a funkcií k nemu prislúchajúcich
  - *Bot* by mal byť schopný vygenerovať jedálniček na určité obdobie dopredu (pár dní, týždeň...), nie iba na jeden deň. Takýmto štýlom si užívateľ môže dopredu pozrieť aké jedlá by si mal dať na nasledujúce

dni a vzhľadom k tomu si môže dopredu nakúpiť ingrediencie a potraviny na navrhované jedlá.

- Funkcia *Nextday* by fungovala vo vylepšenej verzii. Namiesto odpočítania celého prebytku z ďalšieho dňa sa eventuálny prebytok rozpočíta na nasledujúce dni; ak užívateľ zje niečo, čo mu pridá 500kcal naviac, tak *bot* rozpočíta a nageneruje jedálničky najbližších 5 dní tak, že každý deň vytvorí jedálniček o 100kcal menej, než by užívateľ mal prijať, a týmto vytvorí deficit v kalorickom príjme, ktorý sa dorovná prebytočným prijatým kalóriám v jednom dni. Takýto prístup je pre užívateľa aplikácie určite príjemnejší, nakoľko sa nestane to, čo sa nám môže stať teraz. Teraz, v prípade veľkého kalorického nadbytku a zvolenia možnosti *Nextday* aplikácia na ďalší deň vygeneruje jedálniček s málo kalóriami a užívateľ tým pádom bude s veľkou pravdepodobnosťou zažívať pocit hladu. V prípade rozpočítania týchto kalórií na viacero dní sa však tomuto vyhneme.

- Aplikácia by mohla mať funkciu nákupný košík, ktorá by na užívateľom zvolené obdobie vedela nakumulovať dokopy všetky potrebné ingrediencie k vygenerovaným jedlám, aby užívateľ nemal žiadnu námahu s tým, že by musel manuálne prechádzať recepty a pozerať, aké suroviny a ingrediencie potrebuje.
- Mohli by sme si zapísať suroviny a potraviny ktoré máme doma (napr. múka, mäso, vajcia...) a *Guidance bot* sa bude snažiť dávať recepty s využitím primárne dostupných surovín.
- *Guidance bot* by v rámci receptov dokázal detegovať prílohy a eventuálne ich meniť pre zlepšenie nutričných hodnôt a lepšiu flexibilitu, dynamickosť a personalizáciu.

4. Dizajn

V tejto kapitole si rozoberieme dôležité aspekty pri návrhu našej aplikácie. Povedieme diskusiu o spôsoboch riešenia daných dizajnových problémov a v prípade viacerých možností riešenia problému popíšeme aj dôvody, ktoré nás viedli k samotnej voľbe.

4.1 Technológie

V tejto sekcii budeme viesť diskusiu o technológiách, ktoré sme pri práci použili, aké sme mali možnosti a vždy vyvodíme záver s odôvodnením, prečo sme urobili rozhodnutie, aké sme urobili.

4.1.1 Platforma a jazyk

Pre zjednodušenie budeme uvažovať o 3 základných skupinách platforiem, pre ktoré sa robí vývoj v dnešnej dobe:

1. Webová platforma
2. Desktopová platforma
3. Mobilná platforma

Vzhľadom k cieľu našej práce môžeme vyškrtnúť desktopovú platformu. Je to nepraktické a nemalo by to žiaden prínos. Ostala nám mobilná a webová platforma. Dnes sú už obe tieto platformy dosť prepojené a existuje aj viacero technológií, ktoré sa ich snažia spojiť (napríklad *Apache Cordova*[33]). V konečnom dôsledku je však intuitívnejšie spraviť našu prácu ako mobilnú aplikáciu než webovú, kvôli možnosti lepšie demonštrovať výsledky a natívnej podpore technológií ako je GPS alebo fotoaparát, pokiaľ by sme aplikáciu rozširovali o skenovanie čiarového kódu alebo napríklad spomínaného image recognition.

[33] https://cordova.apache.org/

Po rozhodnutí pre ktorú zo základných platforiem budeme tvoriť aplikáciu – mobilná platforma – tak stojíme pred ďalšou úlohou a tou je rozhodnutie aký cieľový operačný systém si vybrať. Momentálne sa na mobilnom trhu nachádzajú v podstate iba dva relevantné operačné systémy, a to sú *Android* a *iOS*. V tomto prípade máme znovu 3 možnosti, ako vytvoriť aplikáciu pre dané OS:

1. Natívna aplikácia pre *Android*
2. Natívna aplikácia pre *iOS*
3. Cross-platform aplikácia pre obe platformy

Ak by sme chceli robiť aplikáciu pre produkciu, tak najrozumnejšie by bolo zvoliť riešenie cross-platform, kvôli tomu, že takto pojmeme oba trhy (jak *iOS* tak *Android*) naraz. V tomto prípade by sme mohli využiť jednu z najznámejších dostupných cross-platform technológií:

1. *Flutter*[34] (*Google*)
2. *Xamarin*[35] (*Microsoft*)
3. *React Native*[36] (*Facebook*)

V našom prípade však chceme ukázať šikovnosť, inovatívnosť a zaujímavosť nápadu, nie cieliť na predaj. Kvôli autorovej znalosti a skúsenosti s jazykom *Java* preto padlo rozhodnutie vybrať si platformu *Android* a vytvoriť tým pádom natívnu androidovú aplikáciu v jazyku *Java*. Alternatívou k jazyku *Java* pri platforme *Android* by mohol byť jazyk *Kotlin*[37], ktorý sa stále viac a viac integruje do programovania pre androidovú platformu odkedy ho *Google* vyhlásil ako oficiálny preferovaný jazyk pre *Android*[38].

---

[34] https://flutter.dev/

[35] https://dotnet.microsoft.com/apps/xamarin

[36] https://reactnative.dev/

[37] https://www.postman.com/

[38] https://techcrunch.com/2019/05/07/kotlin-is-now-googles-preferred-language-for-android-app-development/

4.1.2 Komunikácia s REST API

Pre uľahčenie práce s API využijeme niektorú z už implementovaných knižníc riešiacich túto problematiku. V prípade Javy existuje viacero knižníc na komunikáciu s REST API [39] ktoré môžeme použiť, pričom dve z nich sú najznámejšie:

1. *OkHttp*[40]
2. *Retrofit*[41]

Retrofit je knižnica postavená na *OkHttp*. V našom prípade treba zohľadniť najmä to, že budeme knižnicu využívať na platforme *Android*. V Androide, sieťová komunikácia prebieha asynchrónne [42], preto volíme použitie knižnice *Retrofit*, pretože má lepšiu podporu asynchrónnej sieťovej komunikácie.

4.1.3 JSON parsing

Ďalší z problémov, ktorý budeme musieť v našej práci riešiť je to, v akom formáte budú nami používané dáta a ako ich spracovávať. Obe nami používané služby, ***Spoonacular*** aj ***Nutritionix*** používajú formát JSON[43] (dôvod ich výberu sa dozvieme v ďalšej sekcii Dizajn – Dáta). Pre zjednodušenie práce, rovnako ako pri komunikácii s REST API, aj v tomto prípade použijeme už existujúcu knižnicu. Najpopulárnejšia knižnica na parsovanie formátu JSON v jazyku *Java* je knižnica *gson* [44] od *Google*, ktorá dokáže previesť (serializovať) akýkoľvek POJO [45] do formátu JSON a rovnako dokáže skonštruovať spätne POJO z formátu JSON

---

[39] https://restfulapi.net/

[40] https://square.github.io/okhttp/

[41] https://square.github.io/retrofit/

[42] Takýto dizajn v Androide zvolili vývojári tohto OS z dôvodu, že hlavné vlákno obsluhuje UI, a ak by sa sieťová komunikácia vykonávala na hlavnom vlákne, spôsobí to zamrznutie aplikácie, a tým pádom jej neresponzívnosť, čo nie je žiadúce na mobilnom zariadení.

[43] https://en.wikipedia.org/wiki/JSON

[44] https://github.com/google/gson

[45] https://en.wikipedia.org/wiki/Plain_old_Java_object

(deserializovať). Existuje ešte zopár iných knižníc na parsovanie JSON, tam je však väčšinou potreba riešiť anotácie a iné špecifické konštrukcie pre jazyk *Java*. Rozhranie tejto knižnice považujeme za priamočiare a intuitívne. V našom prípade si vystačíme len s metódami *toJson()* a *fromJson()*. Preto v našej práci použijeme na parsovanie JSON knižnicu *gson*.

4.2 Dáta

Dáta v našej aplikácii využívame na 2 účely. Prvý je manuálne vyhľadávanie a prípadný zápis jedla do aplikácie a druhý je generovanie jedálnička pre užívateľa *Guidance botom*. Napriek zdanlivej spojitosti týchto dvoch činností budeme pristupovať k dátam rozličným spôsobom. Zatiaľ čo na manuálne vyhľadávanie budeme vždy využívať niektorú z API služieb, tak na generovanie jedálničkov pre užívateľa budeme využívať primárne dáta lokálne. Hlavný dôvod rozdielneho spôsobu prístupu k dátam je obmedzenie počtu API volaní na deň pri nami používaných API službách. Preto ak chceme aby naša aplikácia fungovala čo najlepšie s využitím dostupných free API plánov, rozdelíme si prístup k dátam na 2 rozdielne spôsoby a podľa toho spravíme finálny dizajn využívania dát v našej aplikácii.

4.2.1 Dáta pri manuálnom vyhľadávaní jedla

V našej práci na vyhľadávanie jedál použijeme kombináciu dvoch API, a to ***Nutritionix*** a ***Spoonacular***. ***Nutritionix*** je jasná voľba z dôvodu disponovania dátami o reštauráciách a ich menu, ktoré chceme mať v našej aplikácii. Problém je, že ***Nutritionix*** nedisponuje dátami o receptoch, preto musíme na recepty použiť nejaké iné API. Tým pádom nám ostalo rozhodnutie medzi ***Spoonacular*** a ***Edamam***. Obe ponúkané API sú si dosť podobné, no vybrali sme si radšej ***Spoonacular***. To z dôvodu, že ak by sme chceli náš modul **(A4)** rozšíriť o generovanie jedálničkov s častým API volaním (keďže aktuálne budeme využívať primárne lokálne dáta), tak pravdepodobne budeme potrebovať osloviť server viac krát než 5x za minútu, čo je maximálny povolený limit pri ***Edamam*** pre free plán. Napriek tomu, že ***Edamam*** má v súčte omnoho viacej povolených volaní na deň než ***Spoonacular*** (***Edamam*** 5x za

minútu, ***Spoonacular*** 150 za deň)[46], my budeme potrebovať volanie receptov viac a nárazovo, aj keď potom dlho nič. Okrem toho, ***Spoonacular*** má lepšie popísanú API dokumentáciu a celkovo pôsobí dojmom, že ich API je to, na čo sústredia svoj hlavný dôraz a vývoj.

4.2.2 Dáta pri generovaní jedálničkov

Prístup k dátam pri generovaní jedálničkov je v skutočnosti odlišný než pri vyhľadávaní jedla v databáze. Najväčší rozdiel je vo frekvencii potrebných dotazov k dátam v krátkom časovom úseku. Zatiaľ čo pri manuálnom vyhľadávaní je počet API volaní relatívne minimálny, pri generovaní jedálnička potrebujeme omnoho viac dotazov na dáta s receptami, aby sme vygenerovali jedálniček na mieru. Ako presne tieto jedálničky generujeme sa dozvieme ďalej v sekcii Algoritmus. Kvôli veľkému množstvu dotazov na dáta v krátkom časovom úseku padlo rozhodnutie využiť pre generovanie jedálničkov lokálne dáta.

Keďže ***Spoonacular*** poskytuje kategorizáciu receptov podľa chodu, využili sme tejto možnosti, aby generované jedálničky boli zložené z chodov s relevantnými receptami. Spravili sme si teda 3 zoznamy receptov podľa takto dostupných kategórií, a to Breakfast (raňajky), Main course (hlavný chod, čiže recepty na obed a večeru) a Snack (desiata/olovrant). Dáta sme si následne manuálne stiahli s pomocou viacnásobných API volaní na ***Spoonacular*** s využitím softvéru ***Postman***[47], ktorý slúži na testovanie API volaní a vracanie ich odpovedí. Takto získané dáta vo formáte JSON sme si zlepili dohromady, lokálne uložili a môžeme ich využívať na generovanie jedálničkov.

Dokopy sme takto získali 180 receptov na raňajky, 1000 receptov na hlavný chod a 500 receptov na desiatu/olovrant.

---

[46] Tieto čísla korešpondujú s API free plánmi – pri platených API plánoch sú tieto hodnoty diametrálne odlišné, no podľa toho našu aplikáciu v tejto práci nedizajnujeme

[47] https://www.postman.com/

4.3 Architektúra

V tejto sekcii rozoberieme všetky druhy architektúr, ktoré bolo treba navrhnúť pri vytváraní aplikácie. Pri popisovaní jednotlivých architektúr budeme postupovať štýlom zvonku dovnútra. To znamená, že začneme od komplexného návrhu, a postupne sa zanoríme hlbšie do jednotlivých komponent architektúry, ktoré bude treba bližšie rozobrať, zanalyzovať a navrhnúť.

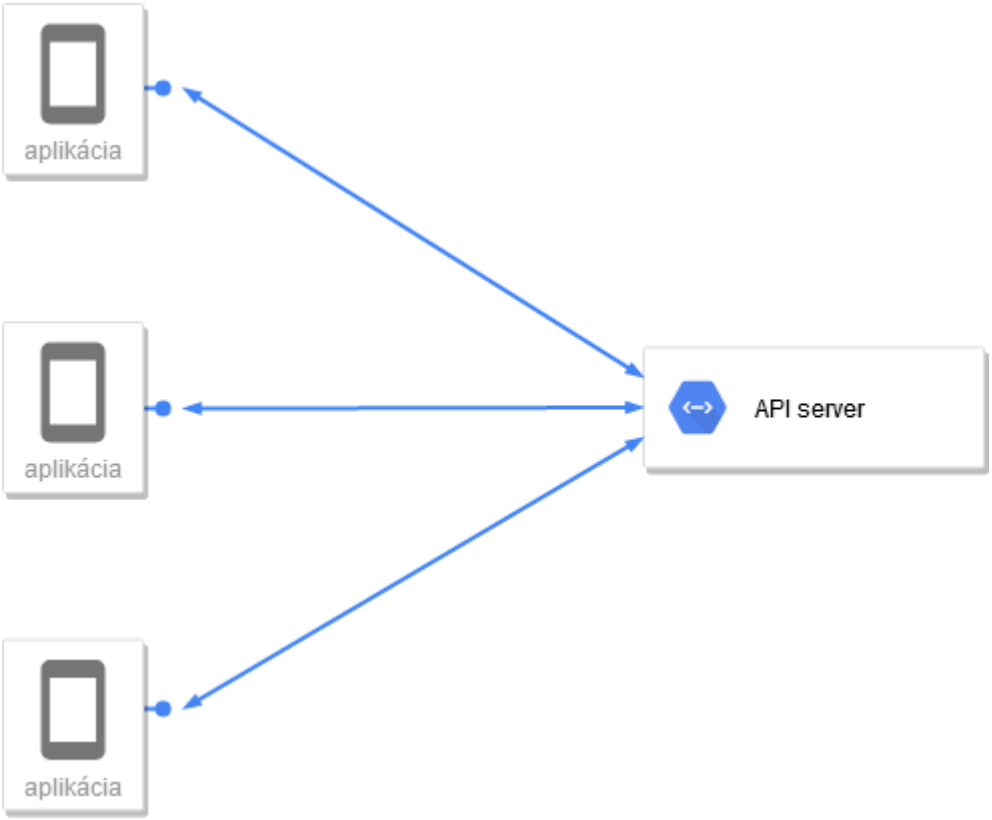
Najprv si povieme o architektúre aplikácie ako celku vo vzťahu k použitému externému API na dáta do našej aplikácie (4.3.1). Potom sa pozrieme na to, ako funguje vnútri samotná aplikácia a uvedieme modulárnu architektúru interného fungovania aplikácie (4.3.2). Ako posledné si rozoberieme architektúru a návrh modulu, ktorý je súčasťou predošlej architektúry, a to modul **(A4)** – *Guidance bot* (4.3.3).

4.3.1 Architektúra pre komunikáciu s externým API

Existuje veľa dizajnových riešení tohto problému komunikácie s externým API. Rozoberieme si dva priamočiare návrhy a povieme si ich charakteristiky. Najprv zadefinujme tri pojmy, a to **aplikácia** - mobilná s ktorou bude interagovať užívateľ, **API server** - server disponujúci databázou, s ktorým budeme komunikovať, keď budeme používať jedno z API na prístup k dátam rozobratým v predošlej kapitole a **medziserver** - nami nakonfigurovaný server, ktorý bude slúžiť ako prostredník medzi aplikáciou a API serverom.

4.3.1.1 Aplikácia – API server

Prvé dizajnové riešenie spočíva v priamom spojení medzi aplikáciou a API serverom. Komunikácia teda medzi nimi prebieha priamo, nejde cez žiadneho prostredníka. Ak užívateľ vykoná v aplikácii nejakú operáciu ktorá potrebuje prístup k databáze, aplikácia posiela API dotaz priamo na API server a server posiela odpoveď rovno aplikácii.



Obrázok 9: Architektúra aplikácia – API server

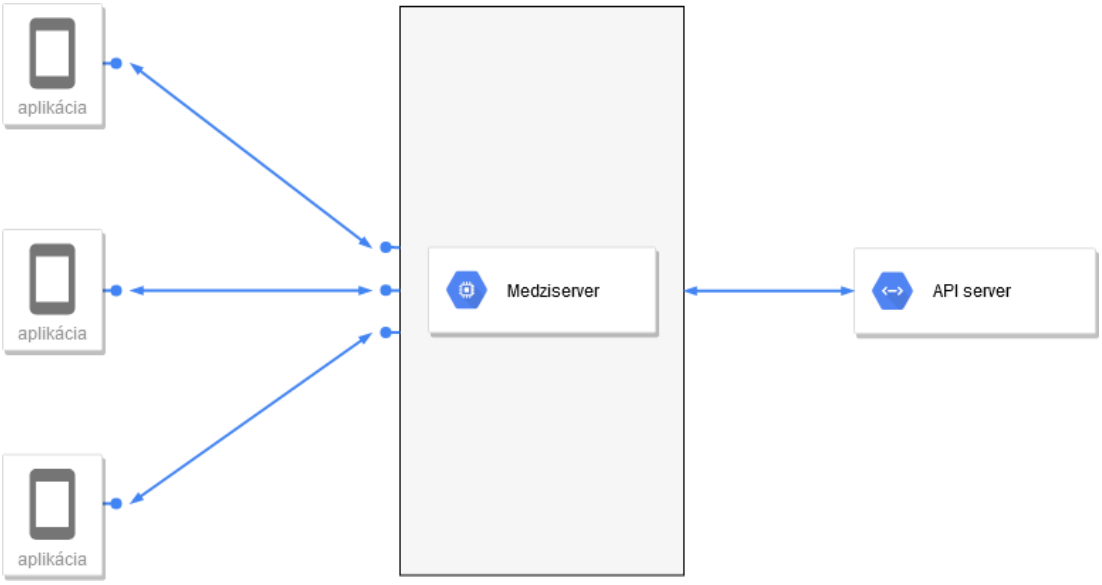
Výhody

- Rýchlejšia odozva (komunikácia), keďže spojenie je priame a nejde cez medziserver
- Aplikácia nie je závislá na funkčnosti medziserveru, takže sa nemôže stať, že ak medziserver vypadne, tak nikomu nefunguje komunikácia s API serverom
- Menej kódu a menej údržby keďže nie je potrebné konfigurovať a ani nijak spravovať medziserver
- Z dlhodobého hľadiska je toto riešenie finančne efektívnejšie, nakoľko nie je potreba vlastniť/prenajímať si žiadne medziservery; všetko beží na koncovom zariadení.
- Veľmi dôležitý faktor tejto architektúry je bezpečnosť. Svoje dáta užívateľ nikam neposiela a všetko má u seba na koncovom zariadení. Nevystavujeme sa zbytočne riziku úniku dát z našich serverov, na ktorých by prebiehali výpočty a vykonávali sa tam algoritmy. Tým pádom vieme jednoduchšie pracovať s myšlienkou nasadiť doporučovací systém pre užívateľa, každému do jeho koncového zariadenia, ktorý by sa učil na základe jeho preferencií a

nebáť sa, že by citlivé údaje o jeho zvyklostiach a návykoch unikli a boli zneužité.

4.3.1.2 Aplikácia – medziserver – API server

Druhé uvažované dizajnové riešenie používa medziserver. V tomto prípade komunikácia medzi aplikáciou a API serverom nejde priamo ale prechádza cez prostredníka.



Obrázok 10: Architektúra aplikácia – medziserver – API server

Výhody

- Výhoda oproti predošlému prístupu je oddelenosť logiky za navrhovaním jedálničkov a samotným UI pre užívateľa. V tomto prípade vieme celú logiku presunúť do medziservera a aplikácia bude slúžiť iba ako front-end pre zobrazovanie. V praxi to znamená, že napríklad modul **(A4)** by bol implementovaný na medziserveri a aplikácia ako taká sa vôbec nemusí starať o jeho chod alebo implementáciu. Vďaka takémuto riešeniu, akékoľvek budúce vylepšenia a opravy modulu **(A4)**, ktoré nemajú dopad na zmenu UI aplikácie, vieme realizovať bez nutnosti aktualizácie aplikácie.
- Keďže sa všetky zložité výpočty dejú na medziserveri a aplikácia robí iba obslužné operácie na zobrazovanie, tak aplikácia nie je nijak (zásadne) limitovaná výpočtovým výkonom koncového zariadenia (smartfónu), ak by sme chceli využívať rôzne náročné výpočtové modely. Problém môže nastať, keď sa preťaží medziserver, no vylepšiť vrstvu na úrovni medziservera ide

jednoduchšie, než užívateľovi nanútiť kúpu drahšieho, modernejšieho a výkonnejšieho zariadenia.

- Nakoľko celá komunikácia je centralizovaná a prebieha cez medziserver, máme lepší prehľad o používaní aplikácie a užívateľoch. Vieme napríklad kešovať API dotazy a tým pádom si budovať lokálnu databázu, a to určite jednoduchšie, než keby si to robila každá aplikácia sama u seba.

4.3.1.3 Verdikt

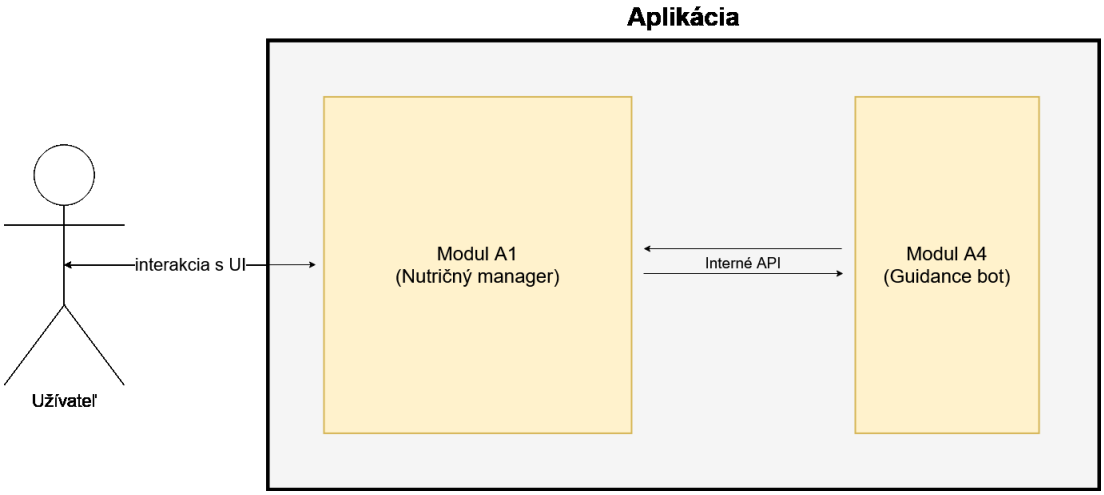
	Apl - API server	Apl - medzi - API			
Odozva	+	-		+	lepšie
Odolnosť voči výpadku	+	-		-	horšie
Veľkosť kódu a údržba	+	-			
Finančné zdroje	+	-			
Bezpečnosť	+	-			
Fyzická distribuovanosť	-	+			
Náročnosť na výkon koncového zariadenia	-	+			
Centralizácia	-	+			

Tabuľka 2: Porovnanie uvažovaných architektúr pre komunikáciu s externým API

Na základe argumentov pre a proti sme sa rozhodli v našej práci použiť prvú spomínanú architektúru nášho projektu. Rozhodli sme sa tak najmä kvôli viacerým výhodám oproti druhej architektúre. Kľúčové faktory hrajúce dôležitú úlohu sú prirodzene bezpečnosť a veľkosť kódu a údržba. V dnešnej dobe sa na mobilných platformách razí trend, aby sa nič čo sa nemusí posielať ani neposielalo a všetko aby mali užívatelia u seba na koncových zariadeniach. Pôsobí to lepším dojmom, pretože užívateľ má väčší pocit bezpečia a súkromia, a zároveň je decentralizácia aj pre nás ako vývojárov menším bezpečnostným rizikom. Veľkosť kódu a údržba je tiež dôležitý faktor, pretože v prípade zvolenia druhej architektúry by sme museli konfigurovať a programovať server, čo by nám zabralo netriviálne množstvo času. Okrem toho je konfigurácia a programovanie serveru niečo, čo nijak priamo nesúvisí s našou prácou a preto je omnoho efektívnejšie to netriviálne množstvo času venovať radšej niečomu týkajúceho sa našej práce priamo. Argument o náročnosti na výkon koncového zariadenia je síce relevantný, no v dnešnej dobe sa už mobilné zariadenia hravo výkonom vyrovnajú stolným počítačom alebo notebookom. Zároveň je nutné dodať, že na koncových zariadeniach nebudeme počítať zložité vedecké výpočty, ale iba heuristiku na vygenerovanie správneho jedálnička, takže nám postačí priemerné mobilné zariadenie.

4.3.2 Architektúra vnútro-aplikačnej komunikácie – interné API

V Špecifikácii sme kládli dôraz na to, aby naša aplikácia bola čo najviac modulárna a aby každá jej časť robila iba to čo má. Toho sa budeme držať aj pri návrhu architektúry vnútorného fungovania aplikácie. Pokúsime sa o rozumné rozdelenie aplikácie na nezávislé moduly, pričom každý bude mať svoju predom určenú rolu v aplikácii. Komunikáciu medzi nimi navrhneme tak, aby bola abstraktná. To znamená, že záležať bude len na vstupe a výstupe pri použitej komunikácii, no na konkrétnej implementácii už nezáleží. Konkrétna implementácia sa tým pádom môže meniť a každý modul sa o ňu stará osobitne. Na takúto komunikáciu si vytvoríme vlastné interné API. Týmto spôsobom dosiahneme väčšiu flexibilitu, rozšíriteľnosť, a modulárnosť aplikácie.



Obrázok 11: Architektúra vnútro-aplikačnej komunikácie

Vnútorné fungovanie aplikácie je založené na vzájomnej komunikácii medzi dvoma hlavnými modulmi v našej práci, a to modulom **(A1)** a modulom **(A4)**. Každý modul má na starosti svoje funkcie, ktoré sme si špecifikovali v kapitole Špecifikácia. V rámci architektúry vnútro-aplikačnej komunikácie sa modul A1 stará primárne o obsluhu UI a interakciu s užívateľom. Modul A4 zasa v tejto architektúre zohráva úlohu v prípade, že užívateľ vykoná takú akciu, ktorá si vyžaduje nejakú z funkcionalít, ktoré implementuje modul A4. Keďže chceme čo najväčšiu nezávislosť týchto dvoch modulov na sebe, vytvoríme si interné abstraktné API na

komunikáciu medzi nimi, pre prípady, kedy je vyžadovaná funkcionalita modulu **(A4)**.

Predtým než nejaké API vytvoríme, si potrebujeme zašpecifikovať kategórie akcií, ktoré môžu v rámci tejto architektúry nastať. Akcie môžeme rozdeliť na 2 základné kategórie:

- Akcie, ktoré nepotrebujú byť spracované modulom **(A4)**
- Akcie, ktoré potrebujú byť spracované modulom **(A4)**

Prvý prípad je z pohľadu architektúry ten jednoduchší. V tomto prípade nie je potrebné iniciovať žiadnu komunikáciu medzi modulmi, pretože danú akciu vykoná a spracuje modul **(A1)** sám, nezávisle od modulu **(A4)**. Medzi takéto akcie patrí napríklad prezretie si nutričných hodnôt vygenerovaného alebo užívateľom pridaného jedla (vygenerované, ako aj manuálne pridané jedlo už máme uložené lokálne, preto nepotrebujeme pristupovať k vzdialenej databáze a pýtať si informácie o jedle), vymazanie jedla ktoré si užívateľ pridal do jedného zo 4 chodov, nastavenie fyzických parametrov užívateľa atď.

Druhý prípad je o niečo zložitejší, pretože je nutné predať vykonanie akcie na spracovanie do modulu **(A4)**. Všetky funkcie, ktoré modul **(A4)** musí implementovať sme si už zadefinovali v Špecifikácii modulu **(A4)**. Na základe tejto špecifikácie rozdelíme tieto funkcie na 3 rôzne akcie, ktoré užívateľ môže vykonať a potrebujeme ich spracovať v tomto module. Tieto akcie, kedy je nutné iniciovať komunikáciu medzi **(A1)** a **(A4)** sú nasledovné:

1. Pridanie jedla do chodu – touto akciou zahrnieme funkcie **A4.C** a **A4.E**
2. Vygenerovanie jedálnička – touto akciou zahrnieme funkcie **A4.A** a **A4.B**
3. Výpočet nutričných limitov z parametrov – touto akciou zahrnieme funkciu **A4.D**

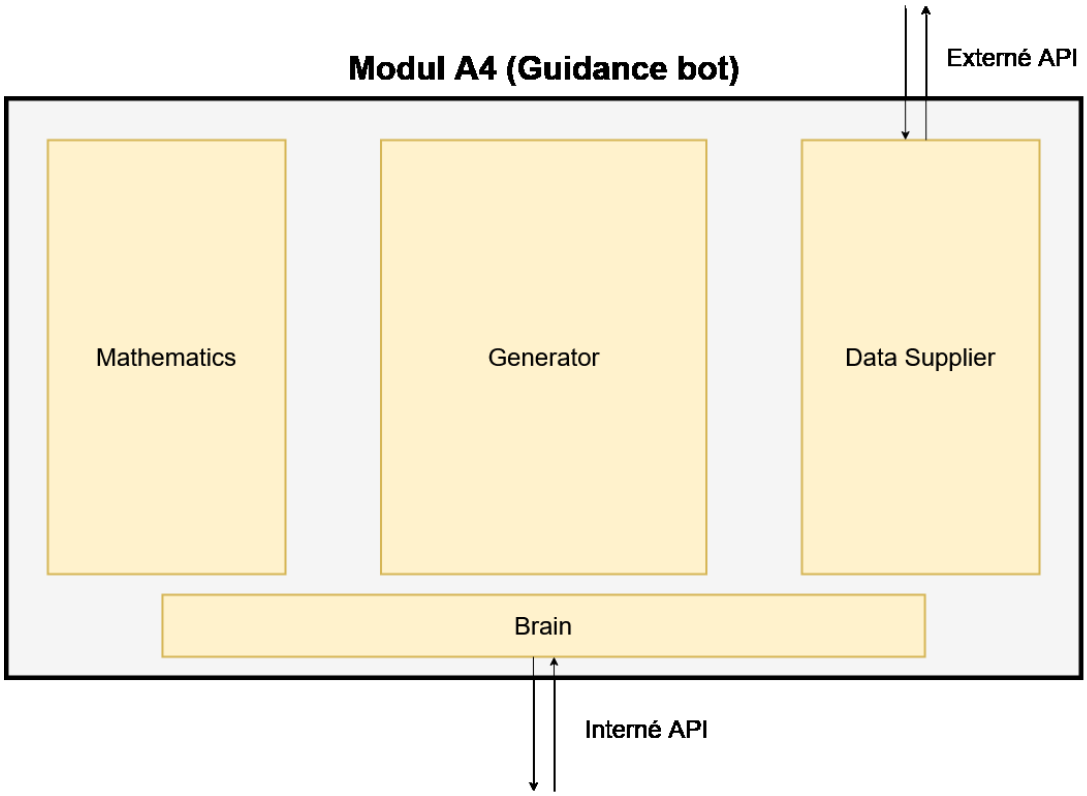
Pre tieto tri akcie si navrhneme interné API. To znamená, že kedykoľvek potrebujeme komunikovať s *botom* kvôli niektorej z týchto akcií, stačí nám iba zavolať metódu, ktorá slúži na jej vykonanie. Z pohľadu **(A1)** nepotrebujeme vedieť akým spôsobom sa vykonávajú dané akcie v **(A4)**; stačí nám, že vieme v akom formáte máme očakávať výsledky. Analogicky, z pohľadu **(A4)** nás nezaujíma, ako spracúva a interpretuje dané odpovede užívateľovi na tieto akcie modul **(A1)**. Z toho nám vyplýva, že spôsob, akým sme chceli navrhnúť túto architektúru sme splnili.

Tieto dva moduly fungujú nezávisle od seba a na prípadnú komunikáciu využívajú predom dohodnuté abstraktné interné API.

4.3.3 Architektúra modulu A4 (*Guidance bot*)

Tento modul je hlavným stavebným prvkom našej práce, a zároveň je to časť, ktorú ide stále zlepšovať. Preto je dôležité spraviť rozumnú architektúru aj vnútri modulu A4. Aj v tomto prípade budeme klásť dôraz na modularitu, flexibilitu vývoja a rozšíriteľnosť v budúcnosti. Modul si tak rozdelíme na menšie logické submoduly, každý podľa svojho účelu. Takýto návrh sme zvolili najmä z dôvodu, aby bolo v budúcnosti jednoduchšie dorábať lepšiu funkcionalitu každému submodulu zvlášť, a aby ich vzájomná závislosť bola čo najmenšia. Niektoré submoduly obsahujú ešte ďalšie menšie bloky, ktoré znovu logicky súvisia so submodulmi v ktorých sa nachádzajú.

Zo Špecifikácie už vieme, aké funkcie má modul **(A4)** implementovať. Tieto funkcie môžeme zhrnúť do 3 rôznych akcií, tak, ako sme si ich zadefinovali v predošlom návrhu. Pre každú z týchto 3 akcií vytvoríme osobitný submodul, ktorý bude mať na starosti vykonanie danej akcie a vrátenie jej výsledku. Okrem 3 submodulov slúžiacich na vykonávanie daných akcií si vytvoríme ešte jeden špeciálny submodul, ktorý bude slúžiť ako rozhranie na komunikáciu medzi modulom **(A1)** a modulom **(A4)**.



Obrázok 12: Vnútorná architektúra modulu A4

Celý modul pozostáva zo 4 hlavných submodulov:

1. Brain
2. Mathematics
3. Generator
4. Data Supplier

Každý z týchto submodulov má svoju vlastnú doménu pôsobnosti v tejto architektúre. Rozoberieme si každý tento submodul zvlášť, povieme si načo slúži.

4.3.3.1 Brain

Brain je veľmi špecifickým submodulom v tejto architektúre. Kvôli modularite a čo najväčšej nezávislosti jednotlivých submodulov padlo rozhodnutie vytvoriť aj tento submodul a vložiť ho do modulu **(A4)**.

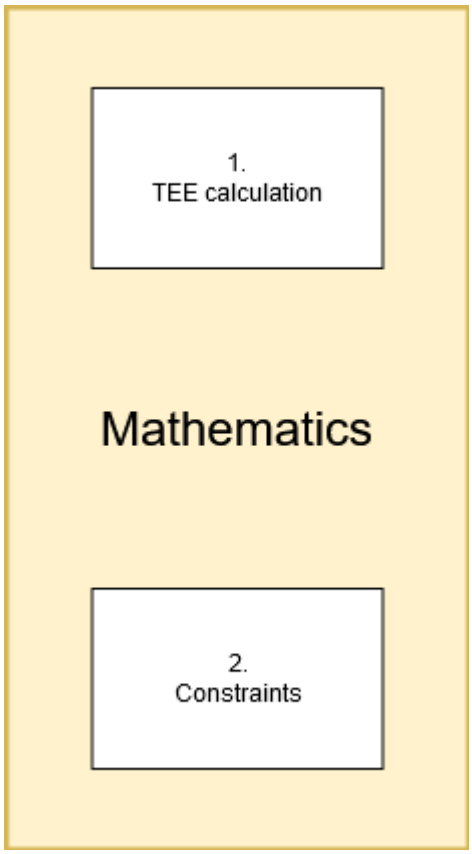
Submodul Brain slúži ako komunikačné rozhranie medzi modulom **(A4)** a zbytkom aplikácie (v našom prípade teda iba modulom **(A1)**, no návrh je taký, že

dodanie iného nového modulu do aplikácie túto architektúru žiadnym spôsobom nenaruší). Týmto návrhom použiť osobitný submodul určený iba na komunikáciu so zbytkom aplikácie odľahčujeme ostatné submoduly v **(A4)** od nutnosti implementovať nejaký spôsob komunikácie s **(A1)**.

V predošlej architektúre sme navrhli používanie interného API pre vnútro-aplikačnú komunikáciu. Je to práve tento submodul, ktorý bude toto API ponúkať. Interné API bude pozostávať z metód, kde každá metóda bude slúžiť na vykonanie a vrátenie výsledku jednej zo spomínaných 3 akcií (aké metódy to sú a ich konkrétnu implementáciu sa dozvieme v ďalšej kapitole Implementácia – Brain). Submodul Brain bude v tomto prípade zabezpečovať korektnú delegáciu akcií správnym submodulom v **(A4)**, aby bola akcia spracovaná tým submodulom, do ktorého pôsobnosti spadá. Zároveň, Brain bude zabezpečovať vrátenie výsledku von z **(A4)**, tiež prostredníctvom interného API.

4.3.3.2 Mathematics

V tomto submodule sa vykonáva výpočet matematického modelu denných limitov nutričných hodnôt a kalórií prispôsobených užívateľovi na základe parametrov. Tieto limity sa následne použijú na zobrazenie a informovanie užívateľa o jeho denných nutričných limitoch a pri generovaní jedálničkov na mieru. Submodul Mathematics teda zabezpečuje spracovanie akcie č. 3.



Obrázok 13: Interný dizajn submodulu Mathematics

Pre výpočet tohto matematického modelu je potrebné vykonať nasledovné:

1. Výpočet TEE[48]
2. Výpočet matematického modelu na základe TEE

Zohľadníme tieto skutočnosti a navrhneme si submodul tak, aby aj tieto 2 úkony boli od seba v rámci submodulu logicky oddelené. Tým pádom si môžeme meniť alebo dodať iný spôsob výpočtu TEE a spôsob stanovovania matematického modelu ostane neporušený a vice versa.

4.3.3.3 Generator

Submodul Generator slúži na generovanie jedálničkov. Zabezpečuje teda spracovanie akcie č. 2. Tento submodul sme nijak špecificky viac nedelili na menšie

[48] Total Energy Expenditure – celkový denný počet kalórií potrebných pre človeka na vykonávanie základných fyzických funkcií ako je dýchanie, cirkulácia krvi, trávenie potravy, cvičenie a iné

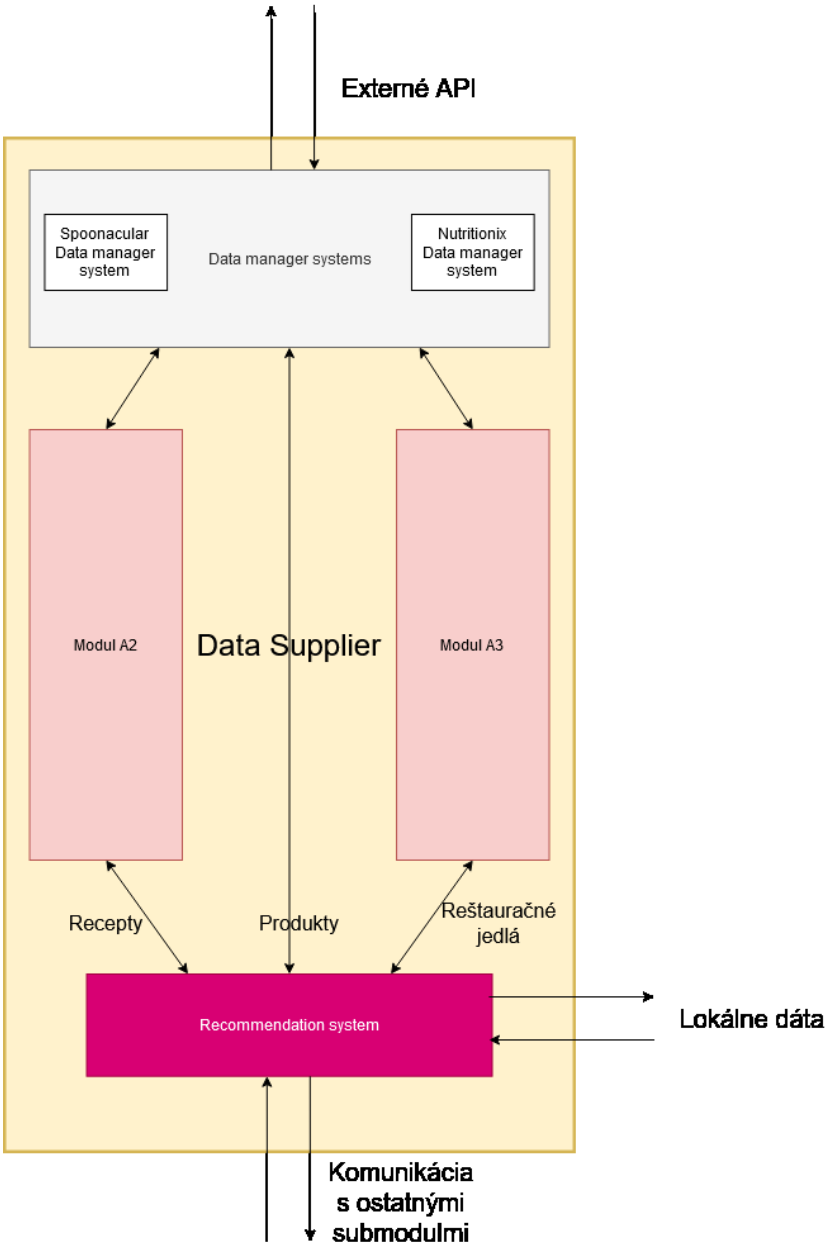
logické jednotky, nakoľko to nebolo nutné. Submodul Generator disponuje jednou (package-private) metódou, ktorá generuje jedálničky na základe parametrov. V prípade, že by sme si chceli dodať nový spôsob generovania jedálničkov (použiť napríklad iný algoritmus na ich generovanie), stačí iba vytvoriť novú metódu a naimplementovať ju. Tento návrh spĺňa všetky naše požiadavky. Je modulárny, flexibilný a dá sa jednoducho rozšíriť, preto nie je potrebné dizajnovať nič sofistikovanejšie.

4.3.3.4 Data Supplier

Submodul Data Supplier slúži na sprostredkovanie dát do celého modulu **(A4)** ako aj do aplikácie. Tento submodul z nami definovaných akcií spracováva akciu č. 1. Pri návrhu tohto submodulu musíme primárne zohľadňovať eventualitu použitia iných dát, než ktoré aktuálne používame. Takáto vlastnosť je pri dobrom návrhu komponenty, ktorá sprostredkúva dáta do aplikácie očakávaná. V našom návrhu budeme rátať s možnosťou rozšírenia aplikácie o personalizovaný doporučovací systém a plné doimplementovanie modulov **(A2)** a **(A3)**, ktoré sa tu budú nachádzať.

V našom návrhu si rozdelíme tento submodul na viacero menších častí:

- Data manager systems
    - konkrétne systémy implementujúce túto časť
- Modul A2
- Modul A3
- Doporučovací systém (Recommendation system)



Obrázok 14: Vnútorná architektúra submodulu Data Supplier.

Základom celého submodulu Data Supplier je časť implementujúca *Data manager systems*. DMS (Data manager system) je časť tohto submodulu, ktorá implementuje komunikáciu s externým API podľa providera. V našej práci máme implementované 2 DMS, a to Nutritionix DMS a Spoonacular DMS, keďže používame API týchto dvoch providerov. Táto architektúra však počíta s tým, že môžeme použiť iné dáta. Na to nám stačí iba vytvoriť a doimplementovať nový DMS, ktorý by komunikoval s iným API. Jediná podmienka, ktorú každý DMS musí spĺňať, tak je správny formát návratových dát.

Z ostatných submodulov v rámci modulu **(A4)** môžeme pristupovať k submodulu Data Supplier s pomocou jeho (package-private) metód, ktoré slúžia ako API pre komunikáciu s externým API. Takýto návrh nám dovolí využívať externé API iba s jednou sadou API v rámci modulu. Toto nám zjednoduší spôsob, akým komunikujeme s externým API a preklad z externého API do API ktoré ponúka Data Supplier necháme čisto na tento submodul. Takému prístupu hovoríme API wrapping[49]. API wrapping nám umožňuje enkapsulovať rôzne druhy externých API volaní do jedného jednotného API, ktoré budeme v našej aplikácii (module **(A4)**) využívať. V praxi to teda znamená, že ak by sme si napríklad chceli dať zobraziť všetky recepty s nejakým parametrom, tak na to máme jednu metódu ktorá nám tie recepty vráti. Neriešime už však ale to, ktoré DMS v skutočnosti obsluhuje požiadavku na dodanie zoznamu receptov. Toto je už úlohou tohto submodulu, aby tento API wrapper používal správne externé API volania. Vďaka tomu môžeme jednoducho meniť dáta ktoré používame v našej aplikácii. Toto využijeme napríklad v prípade generovania jedálničku. Namiesto zavolania externého API použijeme lokálne dáta, no z pohľadu mimo Data Supplier sa nám dáta javia úplne rovnako, ako keby sme využili externé API.

Moduly A2, A3 a Recommendation systém v našej práci síce priamo neimplementujeme, pri vytváraní tejto architektúry však počítame s ich existenciou, resp. s možnosťou ich doimplementovania. Predpokladajme teda že existujú, a na konci si povieme rozdiel v návrhu a našej implementácii.

Všetka komunikácia bude prebiehať cez doporučovací systém. Tento systém sa bude na základe správania a preferencií užívateľa učiť, aké jedlá daný užívateľ preferuje. Takto môžeme ponúkať užívateľovi viac personalizované jedálničky, pretože v prípade generovania jedálnička posunieme do submodulu Generator iba také recepty, ktoré doporučovací systém schváli. Doporučovací systém nám teda v našej architektúre slúži ako filtračná jednotka.

Moduly A2 a A3 nám slúžia na dodatočné spracovanie jednotlivých jedál podľa druhu. A2 sa stará o recepty a A3 o reštauračné jedlá. Tieto moduly nám môžu

[49] https://rapidapi.com/blog/api-glossary/api-wrapper/

implementovať všetko to, čo sme si o nich povedali v Špecifikácia – A2 a Špecifikácia – A3.

Z tejto architektúry v našej aplikácii implementujeme iba DMS. To znamená, že Data Supplier enkapsuluje ako API wrapper iba DMS, a dotazy na DMS spoločne s jeho odpoveďami nijak viac neupravuje. Pre účely demonštrácie funkčnosti *Guidance bota* to stačí. V prípade rozširovania tejto práce je tu však priestor na to, aby sa moduly A2, A3 a doporučovací systém dali ľahko doimplementovať a jednoducho integrovať do existujúcej implementácie. To sme touto architektúrou chceli dosiahnuť.

4.4 Algoritmus

Zvoliť správny algoritmus na riešenie problému generovania jedálničku je jedným z rozhodujúcich faktorov pre dosahovanie plnohodnotných výsledkov tejto aplikácie a naplnenia jej účelu. Musíme zohľadniť najmä 3 rozhodujúce kritériá pri výbere a použití algoritmu:

1. Rýchlosť
2. Správnosť
3. Rozmanitosť

4.4.1 Rýchlosť algoritmu

Rýchlosť algoritmu je prvý dôležité kritérium v prípade výberu algoritmu. Určite nechceme aby algoritmus bežal veľmi dlho, pretože potom by užívateľ pravdepodobne nemal záujem používať našu aplikáciu. Nejaký čas v podobe niekoľko sekúnd navyše, ktorý je užívateľ generovaniu jedálničku ochotný venovať, si vieme získať v aplikácii použitím *loading baru*. Použitie tejto komponenty pôsobí psychologicky lepšie na užívateľa, než keby nám iba zamrzla naša aplikácia počas toho, čo bude generovať jedálniček. V prípade *loading baru* predpokladáme, že bude užívateľ ochotný venovať čakaniu na výsledok viac času[50]. To nám pomôže v tom, že náš algoritmus nemusí dobehnúť rádovo v milisekundách. Čo sa však nemení je

[50] https://iconshots.com/developer/how-progress-bars-or-loaders-impacts-user-experience/

fakt, že algoritmus musí dôjsť k správnemu výsledku rádovo v sekundách – desiatky sekúnd sú už veľa.

4.4.2 Správnosť algoritmu

Druhé kritérium ktoré potrebujeme zaistiť je **správnosť algoritmu**. Algoritmus po dobehnutí musí dávať také výsledky vygenerovaného jedálničku, aby jednotlivé kalórie a makronutrienty v sumách daných jedál spĺňali matematický model. V tomto sme si zase uľahčili vytvorením 10% miery chyby v matematickom modeli, pretože sa stačí zmestiť do intervalov daných nerovnicami podmienok a nie dosiahnuť konkrétne hodnoty.

4.4.3 Rozmanitosť algoritmu

Posledné tretie kritérium nutné zohľadnenia je **rozmanitosť** vygenerovaného jedálnička. To znamená, že požadujeme, aby algoritmus pri rôznych behoch dával rôzne výsledky. Určite nechceme aby algoritmus vracal stále ten istý výsledok (aj keď korektný), pretože chceme dať užívateľovi možnosť vygenerovať si taký jedálniček, aby mu čo najviac vyhovoval. Toto kritérium vlastne hovorí to, že v našom algoritme nie je správne hľadať optimálne riešenie problému, ale iba suboptimálne riešenie. Hľadaním optimálneho riešenia by sme nedokázali zabezpečiť dostatočnú dávku variability ponúkaných jedál v jedálničku.

4.4.4 Druhy algoritmov

Obecný problém generovania jedálničku je NP-úplný, nakoľko ho môžeme previesť na *multidimenzionálny problém batohu* (MDKP) [2]. To znamená, že optimálny algoritmus riešiaci tento problém ani neexistuje. Na vyriešenie problému preto musíme použiť nejaké formy heuristiky, prípadne si problém obmedziť alebo zjednodušiť.

Prvý krok ktorým si zjednodušíme problém generovania jedálničku bude zafixovanie počtu vygenerovaných jedál do jedného chodu na 1. To znamená, že do každého z chodov ktoré máme môžeme použiť iba 1 recept namiesto ľubovoľného počtu. Na takto zjednodušený problém už určite dokážeme navrhnúť polynomiálny algoritmus s časovou zložitosťou $O(b * m^2 * s)$, kde b je počet receptov v dátach na

raňajky, m je počet receptov v dátach na hlavný chod a s je počet receptov v dátach na olovrant/desiatu (snack).

4.4.4.1 Bruteforce

Prvý, najintuitívnejší algoritmus ktorý by sme mohli použiť je **bruteforce**. Bruteforce algoritmus bude skúšať všetky možnosti. V našom prípade máme 4 chody, do ktorých chceme vygenerovať recepty tak, aby spĺňali matematický model. K dispozícii máme 180 receptov na raňajky, 1000 receptov na obed, 1000 receptov na večeru a 500 receptov na snack. To znamená, že na nájdenie výsledku bruteforce algoritmom potrebujeme prejsť prinajhoršom $180 * 1000^2 * 500 = 9 * 10^{10}$ možností. To je veľa. Zároveň, keď berieme v úvahu podmienky ktoré si stanovíme v matematickom modeli, môže sa nám veľmi veľa krát stať, že budeme počítať také kombinácie jedál, kde už pri použití iba jedného jedla do nesprávneho chodu spôsobíme, že daný jedálniček nemôže byť vygenerovaný. Predstavme si situáciu, že si zvolíme na raňajky niečo, čo určite nebude spĺňať limity pre raňajky. To však znamená, že spolu s týmto jedlom skúsime ešte ďalších $1000^2 * 500 = 5 * 10^8$ kombinácií úplne zbytočne. Nejaká forma heurisitky by v tomto určite dokázala pomôcť, no v tejto forme je tento algoritmus nepoužiteľný a preto ho nepoužijeme.

4.4.4.2 4SUM

Typ problému ktorý sa snažíme vyriešiť sa podobá problému, ktorý rieši **KSUM** algoritmy. KSUM algoritmus rieši problém, či daná množina reálnych čísel o veľkosti n obsahuje k prvkov, ktoré sa sčítajú na 0. Tento algoritmus má časovú zložitosť $O(n^{k-1})$ čiže v našom prípade $O(n^3)$. Tento algoritmus je síce o jeden stupeň polynómu rádovo rýchlejší oproti algoritmu predošlému, no problém v tomto prípade je však ten, že tento algoritmus uvažuje iba jeden rozmer. To znamená, že by sme sa museli obmedziť len na kalórie, čo my ale nechceme. V inom prípade by sme museli riešiť viacdimenzionálne verzie a variácie tohto problému, čo však pri zložitosti $O(n^3)$ nemá veľký význam, pretože si určite nijak nepomôžeme a iba zhoršíme. Táto zložitosť je aj tak stále pomalá.

4.4.4.3 Heuristika

V našom prípade sa pravdepodobne javí ako najlepšia voľba nejaká forma heuristiky. Rozumná voľba heuristiky nám môže uľahčiť implementáciu algoritmu

a zároveň môže podávať ešte lepšie výsledky, než už spomínané algoritmy. Správnosť heuristiky zabezpečíme skontrolovaním súčtov a rozmanitosť heuristiky vieme dosiahnuť veľmi jednoducho s použitím nejakej formy randomizácie. Jediné kritérium, v ktorom by nás heuristika mohla sklamať je rovnako ako vyššie spomínané algoritmy – rýchlosť.

Možné formy heuristiky, ktoré by sme mohli použiť sú napríklad bruteforce s orezávaním, heuristika založená na randomizácii či prípadne nejaké variácie týchto heuristík.

V tejto práci sme sa rozhodli naimplementovať plne randomizovaný algoritmus spoločne s nejakými heuristickými funkciami navyše. Tento algoritmus funguje v skutočnosti veľmi dobre a dáva dostatočne uspokojivé výsledky v uspokojivom čase.

4.4.5 Popis použitého algoritmu

Algoritmus použitý v aplikácii je plne randomizovaný. V praxi to znamená, že pri generovaní jedálnička si pre každý chod vyberie náhodne recept z príslušného zoznamu receptov k danému chodu. Keď takto vyberie 4 recepty, tak skontroluje, či splňujú matematický model a jeho podmienky. V prípade, že našiel správnu kombináciu receptov, tak tieto recepty použije a zostaví z nich jedálniček pre užívateľa. V opačnom prípade algoritmus postup opakuje. Vo väčšine bežných prípadov sa empiricky ukázalo, že tento algoritmus podáva veľmi dobré výsledky. Podľa kritérií ktoré sme si stanovili ako rozhodujúce, tak algoritmus v týchto bežných prípadoch všetky spĺňa. Odpoveď dá do 1-2 sekúnd, vygenerovaný jedálniček spĺňa model a rozmanitosť je zabezpečená náhodou. Problém však nastáva, keď potrebujeme riešiť okrajové prípady (edge-cases). Medzi také okrajové prípady patrí napríklad vygenerovanie jedálnička, ktorý má cez 3000 kcal. Na vyriešenie tohto problému sme použili 2 heuristiky, ktoré nám pomáhajú dosahovať kvalitné výsledky aj v okrajových prípadoch.

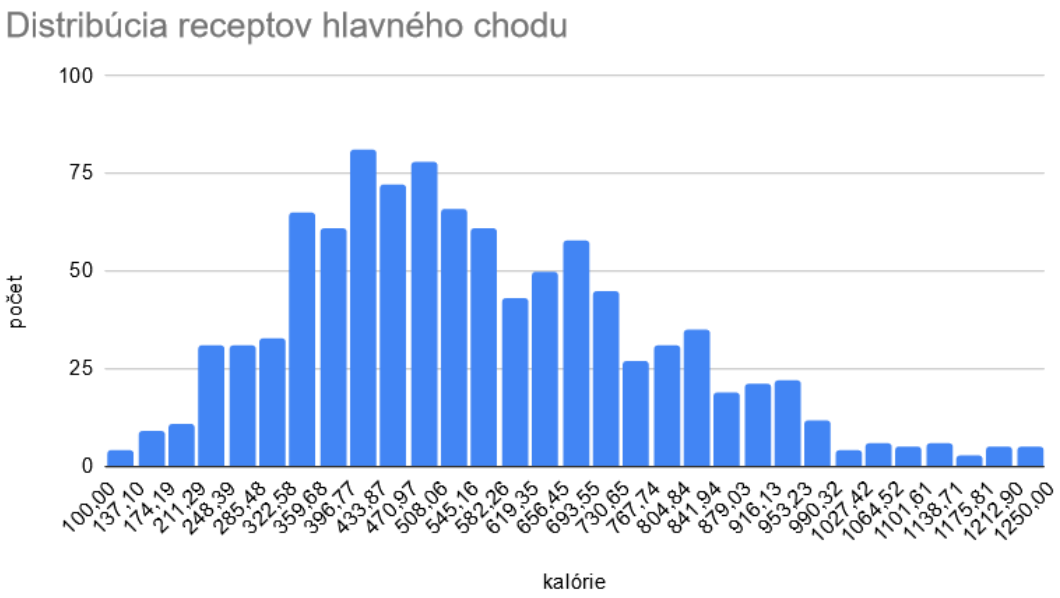
4.4.5.1 Prvá heuristika

Prvá heuristika, ktorú sme do algoritmu implementovali je používanie dvojitých a polovičných porcií. Algoritmus si pri náhodnom výbere receptu okrem toho náhodne zvolí aj veľkosť porcie (normálna, polovičná alebo dvojitá – v princípe

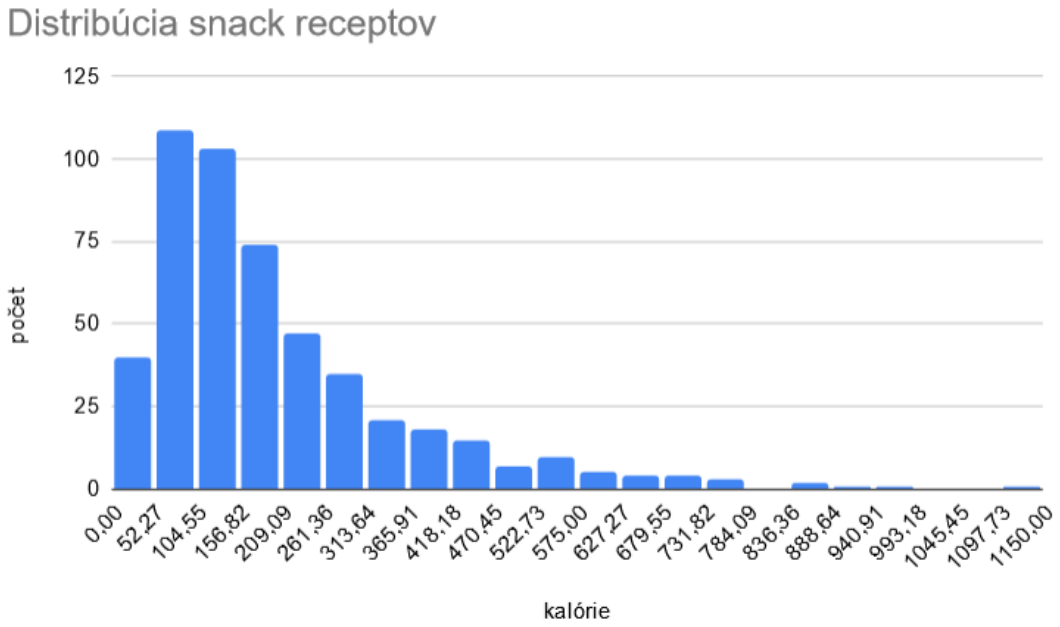
by však nebol problém použiť aj iné násobky). S touto heuristikou sa nám rázom zrýchlil algoritmus pri hľadaní odpovede pre generovanie jedálnička s 3000kcal a viac.

Na nasledujúcich obrázkoch môžeme vidieť rozloženie receptov používaných pri generovaní podľa kalórií. Vidíme, že niektoré hodnoty kalórií majú veľmi malé zastúpenie receptov v daných chodoch. To znamená, že môže nastať situácia, kedy hodnota kalórií, pre ktorú máme vygenerovať jedálniček je daná tak nešťastne, že nevieme nájsť ľahko žiadnu kombináciu receptov, ktorá by okrem celodenných podmienok na príjem makronutrientov spĺňala aj podmienky na rozloženie kalórií do dňa podľa percent. Preto si zavedieme ešte druhú heuristiku, ktorá nám pomôže s vyriešením tohto problému.

Obrázok 15: Graf distribúcie receptov pre raňajky podľa kalórií



Obrázok 16: Graf distribúcie receptov pre obed aj večeru podľa kalórií



Obrázok 17: Graf distribúcie receptov pre desiatu/olovrant podľa kalórií

4.4.5.2 Druhá heuristika

Druhá heuristika, ktorú v našom algoritme využijeme je odignorovanie kalorických podmienok pre chody. Táto heuristika nám zabezpečí to, že nám stačí splniť menej podmienok a tým pádom sa nám zväčší množina prípustných riešení. Táto heuristika však nebude predvolene zapnutá. Najprv sa algoritmus bude snažiť

nájsť riešenie spĺňajúce model za použitia randomizácie a prvej heuristiky. Ak však po istom počte iterácií nenájde riešenie, tak sa algoritmus reštartuje a tentokrát pri hľadaní riešenia použije aj druhú heuristiku.

4.4.5.3 Tretia heuristika

Tretia heuristika je v podstate ultimátum. Táto heuristika vypne všetky obmedzenia až na obmedzenia kalórií. V prípade, že nie sme schopný nájsť riešenie ani za pomoci prvých dvoch heuristík, zapne sa tretia heuristika. Táto heuristika sa použije zvyčajne v situáciách, kedy si užívateľ zvolí také jedlo, ktoré bude mať nejakú makronutričnú hodnotu vysoko nad limity v našom matematickom modeli. V takom prípade by sme nevedeli dodať žiadny jedálniček tak, aby spĺňal náš matematický model vo všetkých ohľadoch. Preto, než aby sme nedodali nič, použijeme radšej túto heuristiku a dodáme užívateľovi jedálniček v takom formáte, aby bol schopný dodržať aspoň kalorické limity (ktoré sú v podstate pri dosahovaní hmotnostných cieľov aj tak stále najdôležitejšie). Táto situácia nám schválne nastane pri demonštrovaní programu v časti Užívateľská príručka – Používanie aplikácie.

5. Implementácia

V tejto kapitole popíšeme implementačné detaily pre plné pochopenie fungovania všetkých dôležitých častí našej aplikácie. Implementačné detaily, ktoré zhodnotíme ako málo dôležité v tejto kapitole spomenuté nebudú. Čitateľ si môže vždy pozrieť kód aplikácie, ktorý je riadne okomentovaný a v ňom sa dozvie všetky ďalšie podrobnosti implementácie.

5.1 Nutričný manager (A1)

Implementáciu nutričného manažéra môžeme rozdeliť do troch hlavných skupín významovo zjednocujúcich implementované triedy, a to:

1. Aktivity
2. Dátové typy jedál
3. Ukladanie dát

5.1.1 Aktivity

Aktivita[51] je na platforme *Android* jednou zo základných komponent, ktoré používame pri vývoji aplikácií. Každá z aktivít v našej aplikácii nejakým spôsobom dáva užívateľovi možnosť interakcie s aplikáciou. To koncepčne podľa vnútro-aplikačnej architektúry radí tieto komponenty do pôsobnosti modulu **(A1)**. Každá aktivita dodržiava pravidlá stanovené v spomínanej architektúre. Ak danú akciu môže vyriešiť sama, tak to urobí, v opačnom prípade kontaktuje *Guidance bota* prostredníctvom interného API. Medzi tie dôležitejšie aktivity ktoré sa hodí osobitne spomenúť patria:

- *MainActivity* – toto je hlavná aktivita v našej aplikácii. Do tejto aktivity sa nám vždy po zapnutí naštartuje aplikácia a z nej sa vieme následne dostať do každej jednej inej naimplementovanej aktivity. Na tejto aktivite máme

[51] https://developer.android.com/reference/android/app/Activity

zobrazený vygenerovaný jedálniček, sem sa nám zapisujú jedlá, vidíme tu celodenný prehľad nutričných hodnôt a iné.

- *FoodAddingActivity* – táto aktivita slúži na manuálne vyhľadávanie jedál. Na zobrazovanie jedál do zoznamu využíva komponentu *RecyclerView*[52] ktorá používa nami naimplementované triedy *FoodAddingAdapter* a *FoodViewHolder*. Toto využijeme pri návrhu a implementovaní dátových typov jedál.
- (*Product/Recipe/RestaurantFood*)*OverviewActivity* – tieto aktivity slúžia na zobrazenie detailných informácií o jedle, ktoré si užívateľ zvolí a klikne naňho pri manuálnom vyberaní.

5.1.2 Dátové typy jedál

V aplikácii pracujeme s 3 druhmi jedál:

- Produkty
- Recepty
- Reštauračné jedlá

Recepty a reštauračné jedlá vypovedajú z názvov o čo sa jedná; medzi produkty zaradzujeme všetko ostatné čo nemôžeme kategorizovať ako recept ani ako reštauračné jedlo, tj. suroviny, polotovary atď. Keďže s jedlom pracujeme v našej aplikácii často, hodí sa nám mať na tieto druhy jedál osobitné špecifické triedy.

Základné triedy pre typ jedlo budú dve triedy – *Food* a *FoodAdapterType*. Obe triedy sú abstraktné a implementujú ich triedy prislúchajúce konkrétnym druhom jedál. Dve typy abstraktných tried máme z dôvodu rozdielneho použitia v aplikácii.

FoodAdapterType a jej implementácie využívame pri naplnení *FoodAddingAdapter* pre zoznam jedál pri manuálnom vyhľadávaní v aktivite *FoodAddingActivity.* V prípade zoznamu nepotrebujeme všetky dostupné informácie o danom jedle (napríklad nutričné hodnoty, váhu jedla a podobne). Stačí nám disponovať iba informáciami, ktoré sú potrebné na zobrazenie miniatúry jedla do zoznamu (názov jedla, URL obrázku miniatúry, kalórie...). Týmto sa šetria zdroje

[52] https://developer.android.com/reference/androidx/recyclerview/widget/RecyclerView

v aplikácii. Získať plné informácie o danom jedle nám stačí až po kliknutí na jedlo, kedy si užívateľ chce prezrieť detaily daného jedla a otvorí sa nám patričná *OverviewActivity*. Tam si už tieto informácie uložíme do niektorej z implementácií triedy *Food*.

5.1.3 Ukladanie dát

Aby bola aplikácia rozumne použiteľná, musíme implementovať nejaký spôsob uloženia dát a ich perzistentnosť.

Na ukladanie dát v aplikácii sme si navrhli a implementovali triedu *DataHolder*, ktorá obsahuje všetky potrebné dáta k fungovaniu aplikácie ako privátne atribúty triedy spoločne s ich *getAttribute()* a *setAttribute()* metódami. Triedu sme navrhli podľa návrhového vzoru *singleton*[53], aby sme mali dáta používané v aplikácii konzistentné za všetkých podmienok a aby sme k ním mali prístup z celej aplikácie. Tento návrh je rovnako prospešný v aplikovaní perzistentnosti dát, pretože máme dáta na jednom mieste.

Perzistentnosť dát znamená to, že nám dáta ostanú v aplikácii aj po zmene konfigurácie[54] (otočenie displeja telefónu, vypnutie aplikácie, prichádzajúci hovor ktorý presunie aplikáciu do pozadia OS...). Napriek dobrému návrhu triedy *DataHolder* by bola aplikácia v podstate nepoužiteľná, ak by sme nemali implementovanú perzistentnosť dát. V takom prípade by po každej zmene konfigurácie došlo k vymazaniu dát z *DataHolder* a nastaveniu predvolených hodnôt atribút.

V OS *Android* existuje pre každú aktivitu tzv. activity lifecyle[55]. V skratke to znamená, že sa aktivita môže nachádzať v rôznych stavoch, a každý tento stav vieme explicitne naprogramovať implementovaním príslušnej callback metódy[56] daného stavu. Pre nás sú v tomto prípade dôležité najmä 2 stavy a to *onCreate* a *onStop*. Stav *onCreate* zavolá svoju callback metódu vždy vtedy, keď je potrebné nanovo

---

[53] https://en.wikipedia.org/wiki/Singleton_pattern

[54] https://developer.android.com/guide/topics/resources/runtime-changes

[55] https://developer.android.com/guide/components/activities/activity-lifecycle

[56] https://en.wikipedia.org/wiki/Callback_%28computer_programming%29

skonštruovať aktivitu. Stav *onStop* zase zavolá svoju callback metódu vždy vtedy, keď sa daná aktivita stane neaktívnou a prejde do pozadia (nevidíme ju už na ploche mobilného telefónu). Jednoduchá úvaha nás privedie k tomu, že by sa perzistentnosť dát dala vyriešiť uložením dát v *onStop()* callback metóde a ich opätovným načítaním v callback metóde *onCreate().* Táto úvaha je správna a uvedieme si náš spôsob implementácie perzistentnosti dát v aplikácii.

Keďže *MainActivity* je hlavná aktivita do ktorej vždy štartuje naša aplikácia, tak v prípade že chceme niečo vykonať, je nutné sa dostať z tejto aktivity do inej. Vždy v prípade, keď zavoláme novú aktivitu z *MainActivity*, tak sa aktivita *MainActivity* dostane do pozadia OS a zavolá callback *onStop().* V prípade akejkoľvek aktivity, kde sa nejakým spôsobom menia dáta aplikácie, sa vždy po uložení dát vrátime späť do *MainActivity.* Preto ak chceme niekde ukladať dáta, tak nám stačí naprogramovať ukladanie dát iba v *onStop()* pre *MainActivity*, pretože máme istotu, že sa nám nikdy dáta v aplikácii nestratia.

Naopak, ak chceme naprogramovať načítavanie dát do aplikácie, to nám už nestačí iba implementovať callback metódu *onCreate()* v aktivite *MainActivity.* To z dôvodu, že aplikáciu môžeme minimalizovať, keď máme otvorenú nejakú inú aktivitu. V takom prípade, pokiaľ má OS dosť zdrojov a našu aplikáciu nezabije, tak po znovuotvorení ostane naša aplikácia na tej aktivite, kde sme používanie aplikácie prerušili. V tomto prípade je potrebné načítavať dáta v callback metóde *onCreate()* každej jednej aktivity. To spravíme jednoduchým spôsobom, že si vytvoríme vlastnú abstraktnú triedu *BaseAbstractActivity* pre všetky aktivity. Táto abstraktná trieda bude dediť od triedy *AppCompatActivity*, z ktorej za normálnych okolností dedí každá jedna aktivita a zároveň bude implementovať callback metódu *onCreate()* na načítavanie uložených dát. Potom každú jednu aktivitu iba nastavíme, aby dedila od tejto abstraktnej triedy.

Samotné ukladanie a načítavanie dát bude fungovať nasledovným spôsobom. Keďže všetky dáta relevantné pre našu aplikáciu máme na jednom mieste, a to je v *singleton* triede *DataHolder,* bolo by užitočné vedieť si perzistentne uložiť nejakým spôsobom rovno celý tento *singleton*. *Android* ponúka možnosť využiť k perzistentnému ukladaniu triedu *SharedPreferences.* Táto trieda ukladá dáta do súboru do pamäte zariadenia a teda je to perzistentný spôsob ukladania dát, odolný voči zmene konfigurácie. Problém je ale ten, že *SharedPreferences* dokáže ukladať

iba jednoduché (v Jave primitívne) dátové typy a typ *String*. Triedu *DataHolder* si uložiť priamo nevieme. Aplikovaním šikovného nápadu však vieme tieto obmedzenia obísť a predsa si uložiť celý *singleton* do perzistentnej pamäte.

Využijeme možnosti knižnice *gson*. *DataHolder* si serializujeme s pomocou *gson* na JSON formát a ten si uložíme prostredníctvom *SharedPreferences* do súboru do pamäte zariadenia ako *String*. Následne, v prípade načítavania zo súboru urobíme opačný postup. Z formátu JSON ktorý máme v súbore uložený ako *String* si deserializujeme spätne náš *singleton* do objektu typu *DataHolder* a uložíme do aplikácie. Tento postup naprogramujeme do vyššie spomínaných callback metód a takto zaručíme perzistentnosť dát v tejto aplikácii.

5.2 Guidance bot (A4)

Spôsob implementácie tohto modulu je veľmi hodnotná informácia, nie len z dôvodu hĺbkového pochopenia fungovania, ale aj z dôvodu znalosti ako rozšíriť tento modul o prípadnú novú alebo inú funkcionalitu. V tejto časti si preto popíšeme všetky submoduly a ich dôležité implementačné detaily.

Celý *Guidance bot* sme si uložili do osobitného package v *Jave*. *Brain* je ako jediný zo submodulov, ktorý má public metódy s ktorými môže komunikovať zvyšok aplikácie. Ostatné submoduly implementujú iba package-private metódy z dôvodu enkapsulácie tohto modulu od zbytku aplikácie. Medzi sebou môžu submoduly komunikovať ľubovoľne, pretože tam na to nemáme žiadne obmedzenie.

5.2.1 Brain

Poznať implementáciu submodulu *Brain* je dôležité najmä z dôvodu, že tento submodul implementuje interné API, ktoré využívame vo vnútro-aplikačnej architektúre. Pre komunikáciu s *Guidance botom* potrebujeme teda poznať aké metódy pre interné API máme k dispozícii. Tie sú nasledovné:

- *public void requestFoodAdapterTypeData(String, int, HashMap, Callback)*
- *public void requestFoodDetailedInfo(FoodAdapterType, Callback)*
- *public void requestRegenerate(List<Boolean>, Context, Callback)*
- *public void requestNHConstraintsCalculation(float caloriesExcess)*

Všetky tieto metódy korešpondujú s akciami, kedy potrebujeme osloviť modul **(A4)**. Prvé dva metódy slúžia na získanie dát ohľadom jedál (či už pri zápise, alebo pri naplňovaní zoznamu) a delegujeme ich do submodulu *Data Supplier*. Tretia metóda sa zavolá ak chceme vygenerovať jedálniček a delegujeme akciu do submodulu *Generator*. Posledná nám vytvorí matematický model a prirodzene, delegujeme akciu do submodulu *Mathematics*. Popis a význam parametrov jednotlivých metód nájdeme v kóde aplikácie.

Tieto metódy majú návratový typ *void* a teda nič nevracajú. Dôvod na takýto návrh metód je ten, že prvé tri metódy potrebujú komunikovať po sieti a posledná metóda nepotrebuje nič vracať (matematický model uloží do *DataHolder*). To, že potrebujú komunikovať po sieti ešte samo o sebe neznamená, že musia byť návratového typu *void*. Problém je však v tom, že sieťovú komunikáciu iniciujeme na OS *Android*. Ako sme už spomínali, v *Androide* prebieha sieťová komunikácia asynchrónne, preto vracať hodnoty z metód prostredníctvom návratového typu nemá význam. V momente vrátenia hodnoty ešte metóda svoje vykonávanie neukončila, keďže komunikuje asynchrónne po sieti a vráti nám niečo, čo nie je nami očakávaný výsledok.

V aplikácii sme preto zvolili spôsob navrátenia hodnôt prostredníctvom callback metód. V momente ako sa dokončí komunikácia po sieti sa zavolá callback a odpoveď sa dostane späť k pôvodnému volaniu.

5.2.2 Data Supplier

5.2.2.1 API wrapper

Data Supplier implementuje API wrapper, ktorý používame v celom module **(A4)**. Na komunikáciu s externým API preto nepotrebujeme poznať ich konkrétnu štruktúru; stačí nám poznať metódy submodulu *Data Supplier* ktoré wrappujú konkrétne API dotazy na externých providerov. Nakoľko v aplikácii pracujeme s 3 druhmi jedál a 2 typmi tried na ich reprezentáciu, *Data Supplier* ponúka metódu na každý jeden druh volania:

- *void requestProductAdapterTypeData(String, HashMap, Callback)*
- *void requestRecipeAdapterTypeData(String, HashMap, Callback)*
- *void requestRestaurantFoodAdapterTypeData(String, HashMap, Callback)*

- *void requestProductDetailedInfo(FoodAdapterType, Callback)*
- *void requestRecipeDetailedInfo(FoodAdapterType, Callback)*
- *void requestRestaurantFoodDetailedInfo(FoodAdapterType, Callback)*

Pri naplňovaní zoznamu zavoláme *AdapterTypeData* druh metód, a ako parametre dáme *String* ktorý bude slúžiť ako kľúčové slovo pre vyhľadávanie takého jedla a *HashMap* s pomocou ktorej ak chceme môžeme filtrovať výsledky. Pri potrebe detailov o danom jedle zavoláme *DetailedInfo* druh metód a ako parameter mu vložíme samotné jedlo, ktoré sme mohli získať z predošlej metódy. Všetky tieto metódy vracajú výsledky prostredníctvom callback.

5.2.2.2 Použitie Data Manager System na implementáciu API wrapper

API wrapper metódy sú v triede *DataSupplier* implementované prostredníctvom konkrétnych DMS. Každá DMS trieda implementuje komunikáciu s externým API, ktoré má priradené. Do wrapper metód tým pádom môžeme iba vložiť metódy konkrétnych DMS objektov podľa toho, ktorý implementuje komunikáciu s ktorým API. V našom konkrétnom prípade máme implementované 2 DMS, jeden pre ***Nutritionix API*** a druhý pre ***Spoonacular API.*** Pri produktoch a reštauráciách voláme metódy *NutritionixDMS,* keďže v našej aplikácii používame na dáta o produktoch a reštauráciách ***Nutritionix API.*** Pri vyhľadávaní receptov resp. generovaní jedálničkov voláme metódy *SpoonacularDMS,* pretože na recepty v našej práci používame ***Spoonacular API***. Keby sme chceli v našej aplikácii používať iné dáta, stačí iba vytvoriť novú DMS triedu pre komunikáciu s novými dátami, naimplementovať ju a potom vložiť metódy z DMS do odpovedajúcej/-ích API wrapper metód.

5.2.2.3 Získavanie dát

Spôsob akým prebieha získavanie dát od *DataSupplier* je nasledovný:

1. Zavolám API wrapper metódu podľa toho, aké dáta potrebujem
2. *DataSupplier* deleguje úlohu konkrétnemu DMS podľa potreby
3. DMS spraví validný API dotaz na API server podľa toho, akého providera má na starosti
4. Do DMS sa vráti odpoveď z API servera v JSON formáte, ktorý si deserializujeme do špeciálneho predom vytvoreného POJO na mieru pre každý jeden API dotaz, ktorý môžeme urobiť z DMS urobiť
5. POJO si s pomocou triedy *PojoConverter* prevedieme do niektorého z datových typov jedla, ktoré používame v aplikácii (dôvod prečo nedeserializujeme JSON rovno do nami používaných typov je ten, že každé API má svoj vlastný formát odpovede, preto potrebujeme každú odpoveď vybaviť unikátne a potom to konvertovať na typ, ktorý používame v aplikácii)
6. Takto vytvorenú odpoveď v známom dátovom type si uložíme do callback metódy a odpoveď sa vracia späť k volajúcemu

5.2.2.4 Lokálne dáta

DataSupplier v našej momentálnej implementácii disponuje aj možnosťou použiť lokálne dáta. Pre túto možnosť sme sa rozhodli kvôli šetreniu API volaní pri generovaní jedálničkov.

5.2.2.5 Reštaurácie

Kvôli demonštrácii schopnosti práce s reštauráciami a jedlami ktoré ponúkajú sme túto funkcionalitu do tejto práce dodali. Avšak, dáta s ktorými pracujeme sú funkčné iba na území USA. Preto nebol dôvod v práci implementovať explicitne synchronizáciu zariadenia s GPS a vyhľadávaním reštauračných jedál v okolí, nakoľko by to tak či tak nefungovalo. Preto sme v práci radšej zvolili prístup, že si môžeme nastaviť súradnice priamo v kóde a simulovať tak to, že sa nachádzame niekde na území USA. Spoločne so súradnicami si takisto v kóde môžeme nastaviť rádius v metroch, do akej vzdialenosti máme reštaurácie uvažovať. Konkrétne si tieto hodnoty môžeme nastaviť v triede *NutritionixDMS*, kde ich máme ako privátne premenné *Pair<String, String> coordinates* a *int radiusMeters*.

5.2.3 Mathematics

V tomto submodule sa dejú všetky výpočty nutné k stanoveniu matematického modelu pre denný príjem nutričných hodnôt a kalórií. Na pomoc s tvorbou matematického modelu sme využili článok pojednávajúci o generovaní jedálničku s pomocou evolučných algoritmov [3]. V článku sa nachádzajú rovnice k tomu, ako správne vypočítať TEE a zároveň tam nájdeme ako zostaviť obmedzenia z nerovníc pre matematický model tak, aby to dávalo zmysel. V našej práci využijeme rovnaký spôsob stanovenia a výpočtu matematického modelu ako bol použitý v tomto článku. Je však možné použiť aj iný spôsob výpočtu matematického modelu keďže nami stanovená architektúra to dovoľuje.

5.2.3.1 Výpočet TEE

Na výpočet TEE budeme potrebovať 3 hodnoty, a to:

- Basal Metabolic Rate (BMR)[57], ktorý sa počíta na základe 4 premenných s použitím Harris-Benedict rovnice [4] na výpočet BMR
  - $BMR_{female} = 10weight + 6.25height - 5age - 161$
  - $BMR_{male} = 10weight + 6.25height - 5age + 5$
- Thermic Effect of Food (TEF) pozostáva z energetického výdaju potrebného na strávenie jedla, čo zodpovedá 10% BMR, ku ktorému musí byť pripočítané
- Thermic Effect of Activity (TEA) je takisto pripočítané k BMR a percentá sa môžu líšiť v závislosti na fyzickej aktivite jedinca podľa tabuľky 3 [5].

Total Energy Expenditure (TEE) zodpovedá odporúčanému dennému príjmu kalórií používateľa. Výpočet je nasledovný:

$$e1: \quad : TEE = BMR + TEF + TEA$$

Stupeň aktivity	Príklad	% z BMR
Sedavý	V posteli	20%
Mierny	Úradník	37,50%
Stredný	Upratovačka, študent	55%
Intenzívny	Robotník, atlét	72,50%

Tabuľka 3: Percentá pridané k BMR v závislosti na fyzickej aktivite jedinca

[57] https://en.wikipedia.org/wiki/Basal_metabolic_rate

V našej práci v implementácii používame rovnaké hodnoty na výpočet TEE. Jediný rozdiel je ten, že môžeme ešte dodatočne zmeniť TEE s pomocou parametru metódy:

- *float getModifiedTEE(float caloriesExcess)*

To sme dovolili, aby sme dokázali reagovať na to, keď užívateľ pri zadaní nenavrhovaného jedla vyberie možnosť *Nextday*. To sa v prípade kalorického nadbytku prejaví práve tak, že tento nadbytok vložíme na ďalší deň ako parameter pri získavaní TEE a tento nadbytok sa od celkového TEE odpočíta.

5.2.3.2 Matematický model

Matematický model stanovíme na základe modifikovaného TEE (pre zjednodušenie budeme aj na modifikované TEE referovať skratkou TEE) a vytvorením obmedzení vo forme nerovníc, tj. lineárneho programu. Tieto obmedzenia sa budú týkať denného príjmu jednotlivých nutričných hodnôt a kalórií rovnako ako aj rozumného rozdelenia príjmu kalórií počas dňa do jednotlivých chodov.

Keďže je prakticky nemožné vytvoriť jedálniček ktorý bude mať presné nutričné hodnoty a kalórie, používame v práci akceptovateľnú mieru chyby 10%. Táto miera chyby sa v implementácii dá zmeniť nastavením premennej *marginError* (v podmienkach budeme referovať na túto premennú skratkou *me*). Aplikovaním tejto miery chyby na jednotlivé potrebné nutričné hodnoty nám vzniknú nasledovné podmienky:

$$p1: \quad (1 - me)TEE \leq calFood \leq (1 + me)TEE$$

kde *calFood* je celkový počet kalórií prijatý jedlom počas dňa. Nasledujúce podmienky nám určujú obmedzenia na príjem jednotlivých makronutrientov. V našej práci je distribúcia kalorického príjmu medzi makronutrienty nasledovná: sacharidy by mali tvoriť medzi 55% až 60%, bielkoviny medzi 10% až 15% a tuky medzi 25% až 30% [6]. Na základe týchto makronutrientov sa počíta energetická hodnota jedál tak, ako to ukazuje Tabuľka 4.

Makronutrienty	Kalórie
Tuky	9 kcal / gram
Sacharidy	4 kcal / gram
Bielkoviny	4 kcal / gram

Tabuľka 4: Podiel kalórií pre jednotlivé makronutrienty

Podmienka *p2* reprezentuje obmedzenie pre denný príjem sacharidov:

$$p2 : (1 - me)carbsReq \leq carbFood \leq (1 + me)carbsReq$$

kde *carbFood* je celkový počet sacharidov prijatý jedlom počas dňa a *carbsReq* je očakávaný denný príjem sacharidov, vypočítaný:

$$e2: \quad : carbsReq = \frac{carb}{100} \times \frac{TEE}{4}$$

pričom *carb* je náhodná premenná z rozsahu [55, 60].

Podmienka *p3* reprezentuje obmedzenie pre denný príjem bielkovín:

$$p3: \quad (1 - me)protsReq \leq protFood \leq (1 + me)protsReq$$

kde *protFood* je celkový počet bielkovín prijatý jedlom počas dňa a *protsReq* je očakávaný denný prijem bielkovín, vypočítaný:

$$e3: \quad : protsReq = \frac{prot}{100} \times \frac{TEE}{4}$$

pričom *prot* je náhodná premenná z rozsahu [10, 15].

Podmienka *p4* reprezentuje obmedzenie pre denný príjem tukov:

$$p4: \quad (1 - me)fatsReq \leq fatFood \leq (1 + me)fatsReq$$

kde *fatFood* je celkový počet tukov prijatý jedlom počas dňa a *fatsReq* je očakávaný denný príjem tukov, vypočítaný:

$$e4: \quad fatsReq = \frac{fat}{100} \times \frac{TEE}{9}$$

pričom *fat* je náhodná premenná z rozsahu [25, 30].

Okrem podmienok určujúce podiel jednotlivých makronutrientov na celkovom kalorickom príjme si určíme ešte podmienky na distribúciu denného kalorického príjmu do jednotlivých chodov. V našej práci sme si určili, že raňajky

budú tvoriť 25% TEE, obed bude tvoriť 35% TEE, večera bude tvoriť 20% TEE a snack bude tvoriť tiež 20% TEE. To si zadefinujeme ako:

- $pBreakfast = 0.25$
- $pLunch = 0.35$
- $pDinner = 0.20$
- $pSnack = 0.20$

Následne s použitím zadefinovaných premenných si zostavíme ďalšie podmienky:

$$p5: \; : (1 - me)pBreakfast \leq calFood_b \leq (1 + me)pBreakfast$$

$$p6: \; : (1 - me)pLunch \leq calFood_l \leq (1 + me)pLunch$$

$$p7: \; : (1 - me)pDinner \leq calFood_d \leq (1 + me)pDinner$$

$$p8: \; : (1 - me)pSnack \leq calFood_s \leq (1 + me)pSnack$$

kde $calFood_x$ reprezentuje počet prijatých kalórií jedlom v danom chode, kde x je začiatočné písmeno chodu, ktorý reprezentuje.

Na základe týchto podmienok si v našej implementácii vytvoríme matematický model jedálnička ktorý budeme používať pri generovaní jedálnička.

Všetky tieto podmienky máme v kóde implementované ako typ *Pair<Float, Float>,* kde prvá hodnota objektu je spodná hranica a druhá hodnota objektu je horná hranica nerovnice.

5.2.4 Generator

Submodul *Generator* v tejto práci implementuje jednu metódu ktorá generuje jedálniček:

- *void randomizedFoodGeneration(List, Context, boolean, Callback)*

a k nej pomocné privátne metódy. Pri požiadavke použiť iný spôsob generovania jedálničku stačí iba vytvoriť novú (package-private) metódu a naimplementovať ju tak, aby používala korektný *Callback* na navrátenie jedálničku v správnom formáte. V nasledujúcich podkapitolách si popíšeme spôsob, akým funguje metóda *randomizedFoodGeneration()* ktorá generuje jedálničky v tejto aplikácii.

5.2.4.1 Dáta použité pri generovaní jedálničkov

To, prečo sme sa rozhodli na generovanie jedálničkov radšej použiť lokálne dáta sme už rozobrali v časti Dáta pri generovaní jedálničkov. Tieto dáta máme uložené v súbore *assets* v projekte tejto aplikácie v troch súboroch formátu JSON, a to *breakfast_recipes.json*, *main_course_recipes.json* a *snack_recipes.json*. Každý tento súbor obsahuje recepty podľa svojej kategórie, ako napovedajú ich názvy. Tieto dáta obsahujú iba tie najdôležitejšie a nutné informácie o receptoch, ktoré potrebujeme pri výbere vhodných receptov na zostavenie jedálnička, a to sú:

- názov
- id
- URL obrázku jedla
- nutričné hodnoty
    - kalórie
    - tuky
    - sacharidy
    - bielkoviny

Tieto informácie nám stačia k tomu, aby sme mohli implementovať algoritmus, ktorý bude generovať jedálničky.

6. Užívateľská príručka

V tejto kapitole bude popísané ako pracovať s aplikáciou z užívateľskej perspektívy. Prvá časť popisuje ako aplikáciu nainštalovať resp. nakonfigurovať aby sme ju mohli používať a otestovať a v druhej časti si povieme ako danú aplikáciu ovládať.

6.1 Konfigurácia

Keďže aplikácia nebola zverejnená na *Google Play,* je potrebné k jej spusteniu disponovať **Android Studio IDE**. Stiahnuť Android Studio sa dá na tejto stránke: https://developer.android.com/studio/archive. Aplikácia bola vyvíjaná v Android Studio verzia 3.6.3, preto je doporučené použiť rovnakú verziu IDE, pre zaistenie rovnakých podmienok. Na stránke je možnosť stiahnuť danú verziu IDE podľa používaného OS.

Spoločne s Android Studio je potrebné mať aj zariadenie, na ktorom môžeme aplikáciu spustiť. Sú dve možnosti ako to urobiť:

1. Fyzické zariadenie Android s verziou aspoň 6.0 (API 23)
2. Android emulátor

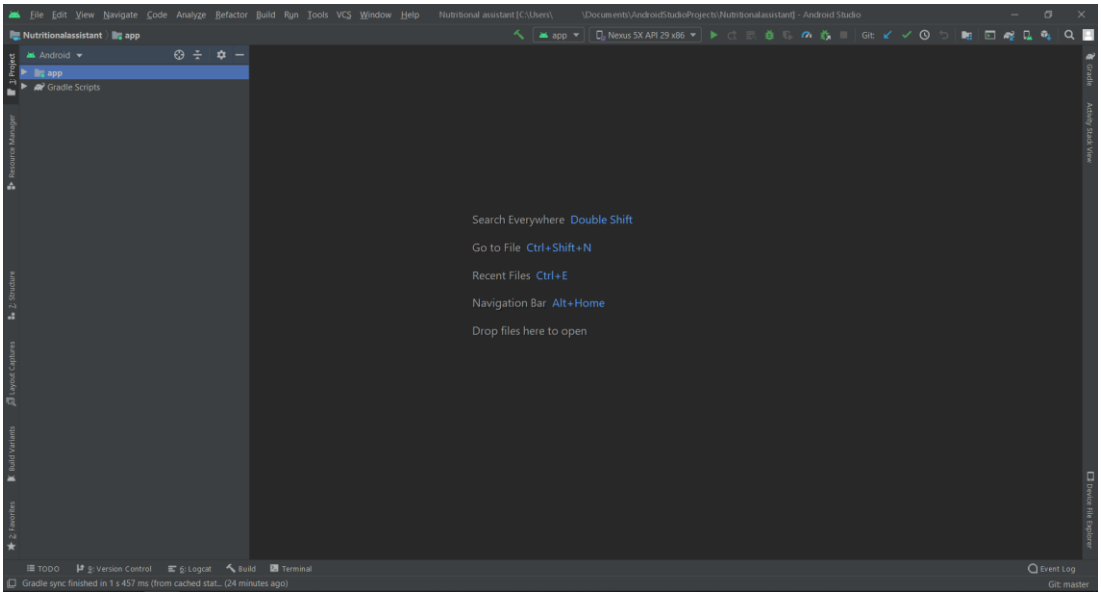
Pre spustenie na fyzickom zariadení je potrebné si najprv nastaviť telefón do stavu „Možnosti pre vývojárov“ (Developer mode) a v ňom potom povoliť možnosť „Ladenie USB“ (USB Debugging). Spôsob povolenia tohto stavu sa môže líšiť zariadenie od zariadenia, ale väčšinou funguje metóda, ktorá je popísaná na tejto stránke.

V prípade emulátoru, ak sme predtým nedisponovali Android Studiom, tak emulátor sa dá nainštalovať spoločne s IDE počas inštalácie.

Počas vývoja boli používané obidve možnosti na testovanie aplikácie. Ako fyzické zariadenie na testovanie bol používaný mobilný telefón **Huawei P8 Lite** (číslo modelu ALE-L21 s verziou Android 6.0) a ako emulátor v Android Studio bol používaný **Nexus 5X API 29 x86.**

V priloženom zip archíve nájdeme projekt aplikácie, ktorý si rozbalíme a následne ho otvoríme v Android Studiu. Android Studio vygeneruje všetko ostatné potrebné. Projekt preto pozostáva iba z nutného minima súborov potrebných na skompilovanie aplikácie.

Po otvorení a načítaní projektu by sme sa mali dostať do stavu podobného v nasledujúcom obrázku:

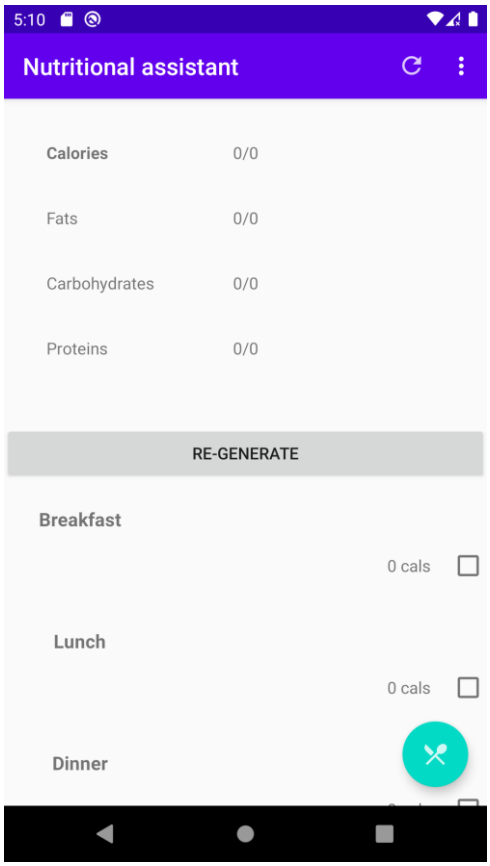


Obrázok 18: Úvodná stránka projektu v Android Studio

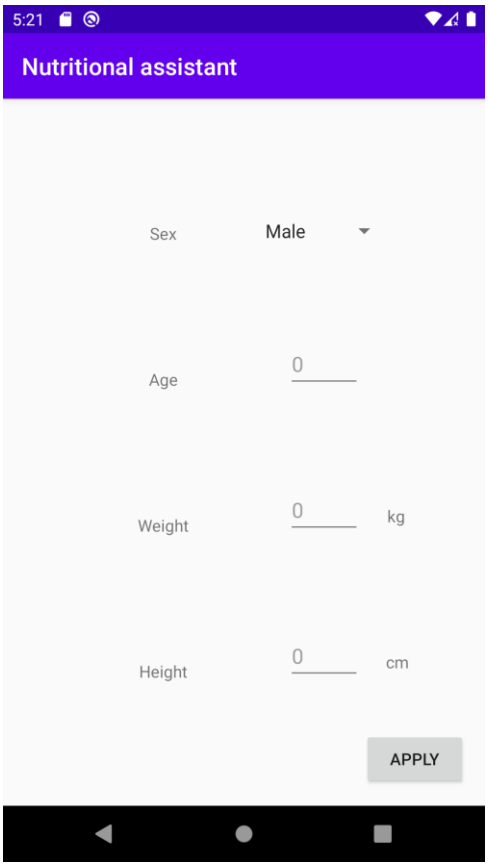
Teraz záleží na tom, aký spôsob otestovania aplikácie sme si vybrali. V prípade, že sme sa rozhodli pre emulátor, tak si v hornej lište z výberu zariadení vyberieme daný emulátor a následne stlačíme zelený trojuholník vedľa pre skompilovanie aplikácie, nainštalovanie do emulátoru a následné spustenie na emulátore. V prípade, že budeme aplikáciu testovať na fyzickom zariadení, je potrebné zariadenie najprv pripojiť s pomocou USB káblu do počítača. Počkáme na zosynchronizovanie s OS a prostredím a potom ho vyberieme, rovnako ako emulátor, z dostupných zariadení. Aplikáciu nainštalujeme a spustíme rovnakým spôsobom (stlačením zeleného trojuholníka) ako emulátor. Po vykonaní týchto krokov by sa nám mala aplikácia spustiť.

6.2 Používanie aplikácie

Po prvom zapnutí aplikácie vyzerá jej obrazovka ako na obrázku 19:



Obrázok 19: Úvodná obrazovka aplikácie



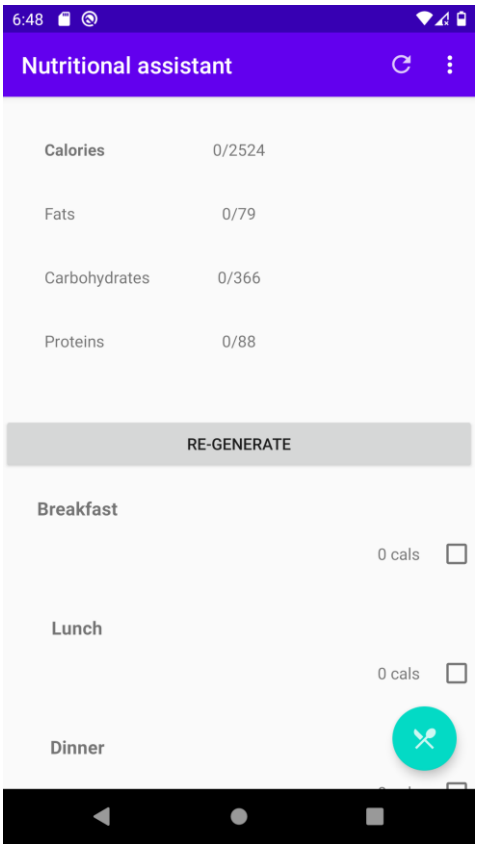
Obrázok 20: Aktivita na nastavenie parametrov

Ako prvé čo musíme urobiť, aby sme mohli aplikáciu plnohodnotne využívať je nastavenie našich fyzických parametrov. Na to klikneme na možnosti hore vpravo (tri bodky) a vyberieme možnosť *User parameters*. To nás dostane na aktivitu na obrázku 20, kde si vyplníme parametre a kliknutím na tlačidlo *APPLY* ich potvrdíme. To nás dostane na ďalšiu sériu aktivít.

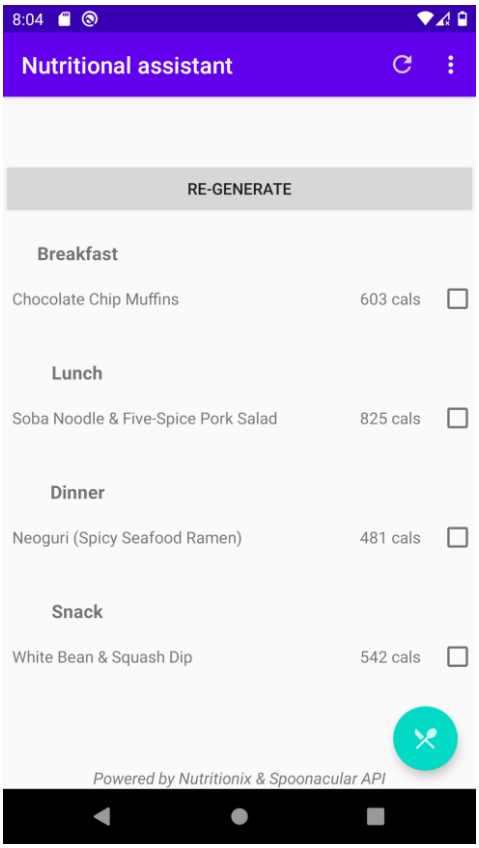
Obrázok 231: Aktivita výberu životného štýlu

Obrázok 242: Aktivita výberu hmotnostného cieľu

Na prvej aktivite si vyberieme možnosť ako chceme naše nutričné hodnoty nastaviť (automaticky / manuálne). Ak si vyberieme manuálne, tak nás to dostane na aktivitu, kde si musíme sami manuálne vyplniť limity nutričných hodnôt. Väčšina užívateľov na ktorých cielime našu aplikáciu však pravdepodobne nebude vedieť, aké nutričné hodnoty by mali mať, preto si aj my v tejto demonštrácii vyberieme možnosť automatického nastavenia. Toto nastavenie nás dostane na ďalšiu aktivitu zobrazenú na obrázku 21, kde si vyberieme z dostupných možností náš životný štýl, pričom máme aj dostupné informácie o tom, čo jednotlivé životné štýly znamenajú. Po výbere životného štýlu si ešte vyberieme náš hmotnostný cieľ, a to teda či chceme schudnúť, pribrať alebo udržať svoju hmotnosť (Obrázok 22). Po výbere aj tohto parametra nás aplikácia vráti späť na úvodnú aktivitu, spoločne s vypočítanými nutričnými hodnotami, ktoré máme stanovené na deň (Obrázok 23). V tejto demonštrácii sme zvolili miernu aktivitu (Mild activity) a ako cieľ sme si zvolili schudnúť (Lose).



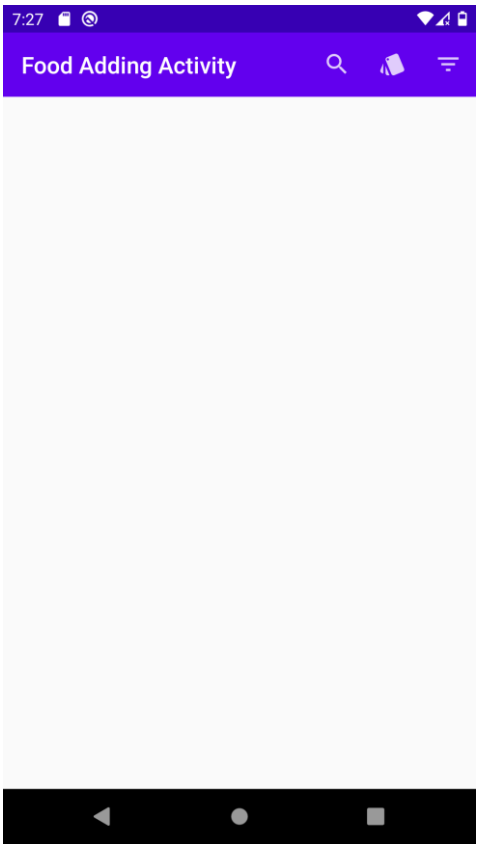
Obrázok 23: Úvodná aktivita po nastavení parametrov



Obrázok 24: Úvodná aktivita po vygenerovaní jedálnička

Po tom čo si nastavíme parametre si môžeme nechať vygenerovať jedálniček na mieru stlačením tlačidla *RE-GENERATE*. To nám vygeneruje jedálniček, ktorý keď budeme dodržiavať, tak splní naše denné nutričné limity (splní náš vytvorený matematický model). Detaily jedál si môžeme zobraziť kliknutím na ich názov. Kliknutím na *checkbox* daného vygenerovaného jedla si jedlo zaškrtneme ako zjedené a jeho nutričné hodnoty sa nám prirátajú k aktuálnym nutričným hodnotám. Pokiaľ by sme si chceli jedálniček vygenerovať nanovo, stačí kliknúť znovu na *RE-GENERATE* a aplikácia nám vygeneruje nový jedálniček.

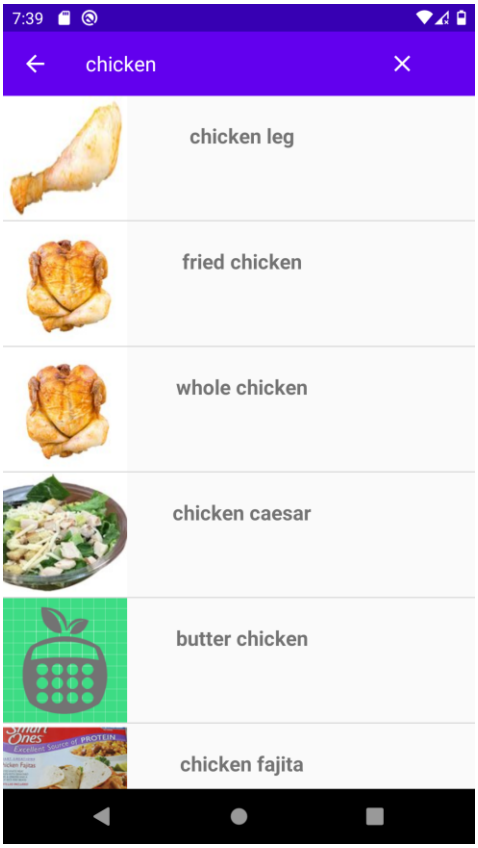
Okrem generovania jedálničkov je naša aplikácia takisto nutričným manažérom. To znamená, že si môžeme zapísať do denného príjmu aj iné než vygenerované jedlá. To urobíme nasledovne:



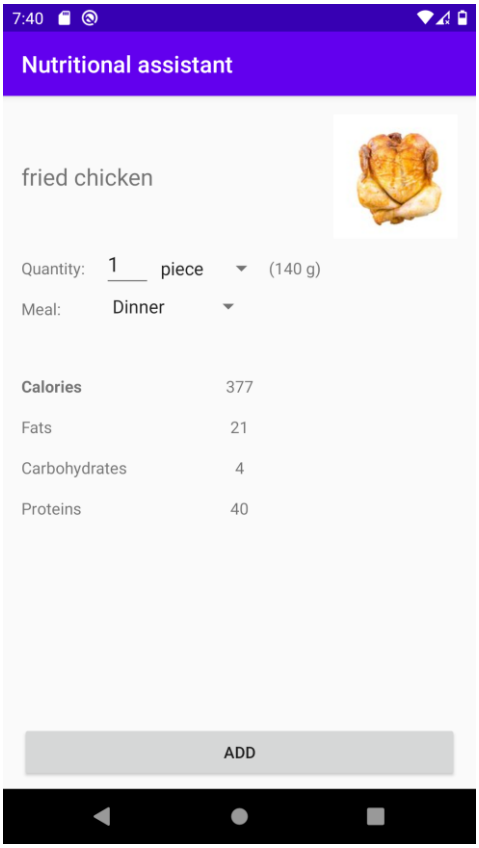
Obrázok 25: Aktivita na vyhľadávanie a pridávanie jedla

Kliknutím na floating button (zelené tlačidlo vpravo dole) sa nám otvorí nová aktivita, kde môžeme vyhľadávať jedlá. Vysvetlíme si postupne z prava, čo jednotlivé ikony znamenajú. Kliknutím na prúžky sa nám otvorí filter, s pomocou ktorého si môžeme zadať špecifické nutričné požiadavky na vyhľadávané jedlo. Ikona štítkov vedľa slúži na výber kategórie jedla (produkty, recepty, reštauračné jedlá). Nakoniec, intuitívne, lupa slúži na vyhľadávanie. Po nastavení parametrov na vyhľadávanie tak, ako chceme, zadáme kľúčové slovo jedla ktoré chceme vyhľadať a **potvrdíme vyhľadávanie!** Je dôležité vyhľadávanie jedla potvrdiť na klávesnici stlačením tlačidla ENTER/lupy, lebo kvôli šetreniu API volaní nevyskočí zoznam jedál sám automaticky. V demonštrácii sme si zvolili produkty a dali ich vyfiltrovať, aby mali minimum 20g tuku. Ak zvolené parametre nejaké jedlá spĺňajú, tak sa nám zobrazia do zoznamu na obrazovku ako môžeme vidieť na obrázku 26. Pre produkty platí, že máme 2 kategórie produktov a to *Common* a *Branded.* Ak si zvolíme nejaký *branded* produkt, tak uvidíme aj jeho značku. V prípade *common* sa nám jeho značka nezobrazí. Ak nájdeme jedlo ktoré si chceme zapísať, klikneme na neho a zobrazí sa nám nová aktivita, ktorá ponúkne bližšie detaily o danom jedle a takisto ponúkne

možnosť si dané jedlo zapísať s nami určenou váhou. V demonštrácii sme si nechali vyhľadať produkty podľa kľúčového slova *chicken* a následne sme si vybrali produkt *fried chicken*.

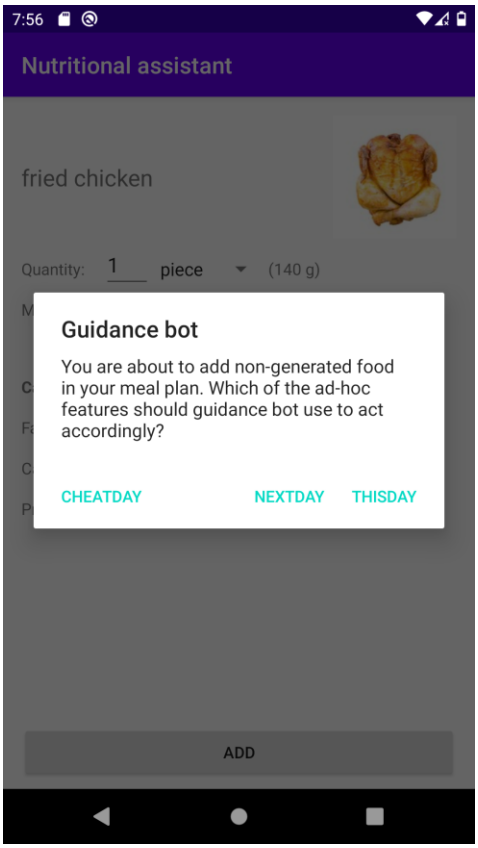


Obrázok 26: Zoznam jedál vo vyhľadávaní

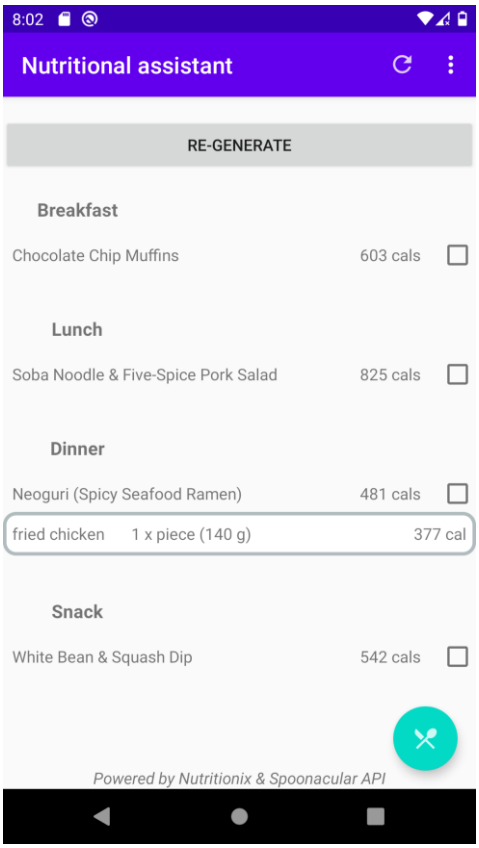


Obrázok 27: Detail o zvolenom produkte

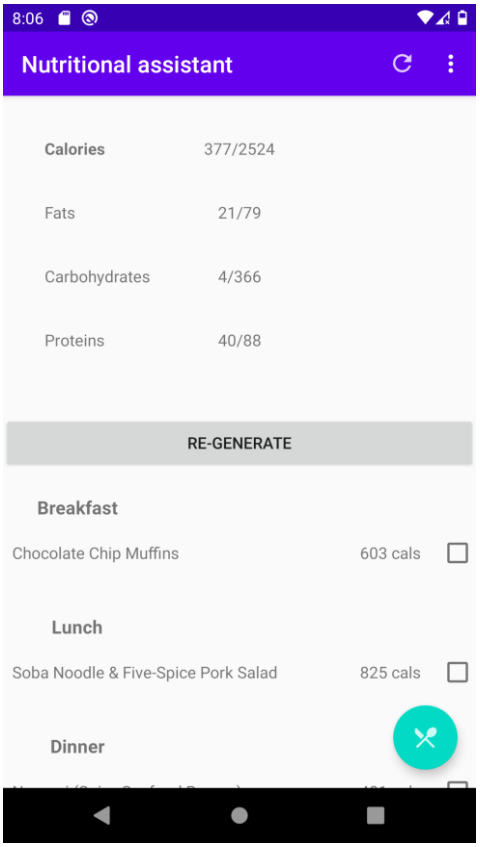
Ak to dané jedlo ponúka, tak si môžeme vybrať z viacerých druhov váhy (kus, gram, unca...) a vybrať si k nej kvantitu. Takisto si môžeme vybrať chod, do ktorého chceme dané jedlo zapísať. Keď sa pokúsime jedlo zapísať (prvý krát počas dňa), tak aplikácia vyhodí upozornenie o tom, že sa snažíme pridať nevygenerované jedlo. To znamená, že je potrebné zvoliť niektorú z ad-hoc funkcií *Guidance bota*, ktoré boli popísané v Špecifikácii – Modul A4. Po zvolení niektorej z ad-hoc funkcií sa zvolené jedlo zapíše do manažéra a aplikácia sa vráti späť na hlavnú aktivitu.



Obrázok 28: Upozornenie na výber ad-hoc funkcie



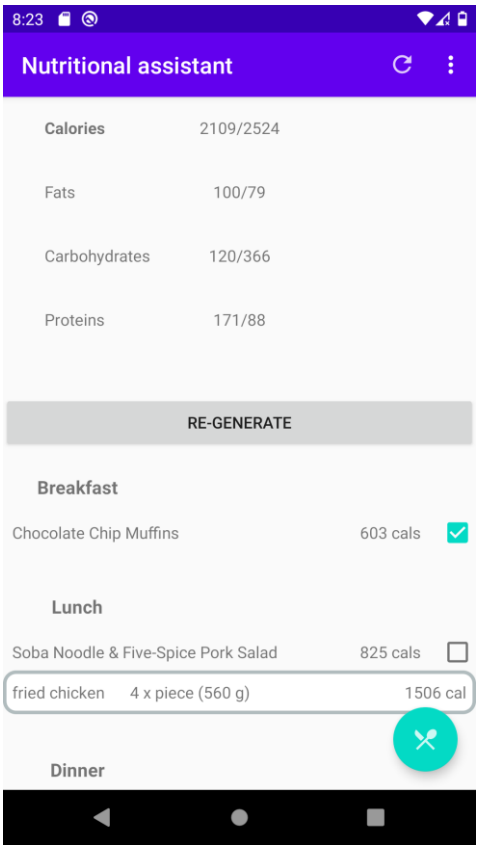
Obrázok 29: Úvodná obrazovka s pridaným jedlom



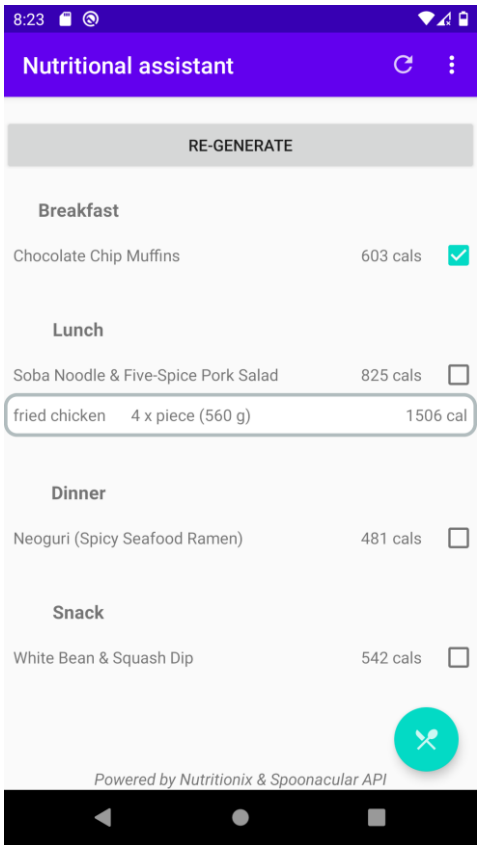
Obrázok 30: Úvodná obrazovka po pridaní jedla – pripočítané jeho nutričné hodnoty

Aj v prípade pridaného jedla si môžeme na dané jedlo kliknúť a zobrazí sa nám jeho detailný popis (či už je to produkt, recept alebo reštauračné jedlo). V prípade, že dané jedlo v manažéri z akéhokoľvek dôvodu už nechceme mať, stačí potiahnuť prstom sprava doľava po ovále v ktorom máme zobrazené jedlo na hlavnej aktivite a jedlo z manažéra zmizne (jeho nutričné hodnoty sa odrátajú z celkových hodnôt).

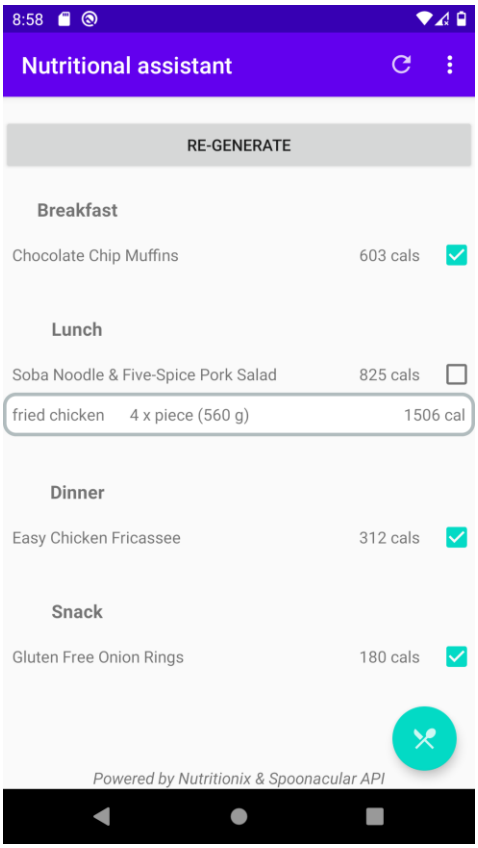
Pre demonštráciu schopnosti pregenerovať jedálniček tak, aby sme boli schopný stále dodržať limity zaškrtnime Breakfast ako zjedené a zadajme väčšie množstvo daného produktu *fried chicken* do niektorého chodu (napríklad Lunch). Po upozornení na zadanie nenavrhovaného jedla si vyberieme možnosť *Thisday* a následne pregenerujeme jedálniček, aby sme ukázali, že vygenerovaný jedálniček bude upravený v reálnom čase tak, aby zohľadňoval situáciu v akej sa nachádzame.



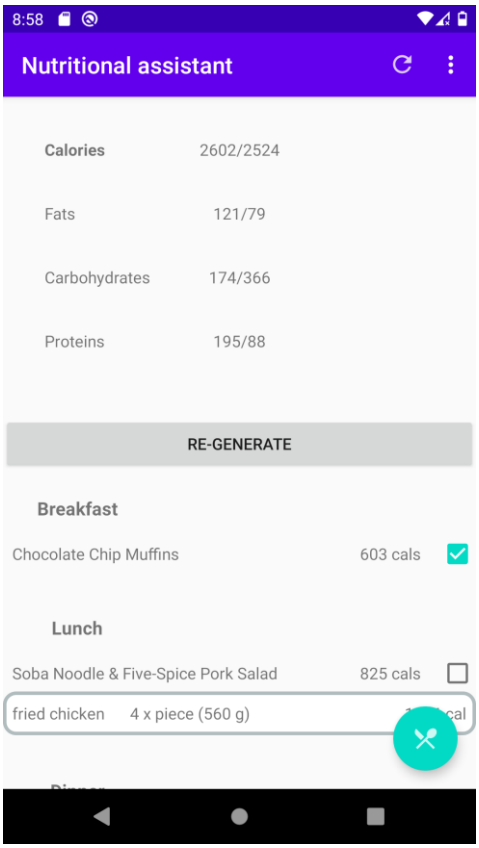
Obrázok 31: Pred generovaním (1)



Obrázok 32: Pred generovaním (2)



Obrázok 33: Po generovaní (1)



Obrázok 34: Po generovaní (2)

Vidíme, že algoritmus našiel ďalšie 2 také recepty, ktoré dokážu čo najlepšie splniť to, aby sme sa zmestili do denného limitu. V tomto prípade nám nedokázal algoritmus poradiť nič lepšie, len nás zmestiť do kalorického obmedzenia, nakoľko ostatné makronutričné hodnoty boli príliš veľké.

7. Záver

Naplnením hlavných cieľov táto práca spĺňa požadované zadanie. Užívateľ má prehľad o svojej strave vďaka implementácii modulu **(A1)** a môže si ju monitorovať. Práca zároveň implementuje adaptívne navrhovanie jedálnička, na čo sme kládli najväčší dôraz pri vypracovávaní a implementovaní tejto práce. K dosahovaniu dobrých a uspokojivých výsledkov aplikácia využíva randomizovaný algoritmus spoločne s 3 druhmi heuristík. V našej práci sme chceli dať užívateľovi takisto možnosť využiť stravovanie v reštauráciách a dynamicky reagovať na takúto formu stravovania. To sa nám v práci podarilo simulovaním našej pozície niekde na území USA a dokážeme vyhľadávať z takto simulovanej pozície jedlá z reštaurácií v našom okolí.

Okrem spomínaných rozšírení pri popise modulov v kapitole Špecifikácia by sa aplikácia dala rozšíriť viacerými spôsobmi. Mohli by sme aplikáciu spraviť v niektorej zo spomínaných cross-platformových technológií aby sme dokázali využívať aplikáciu aj na *iOS* zariadeniach. Ďalšia možnosť vylepšenia tejto aplikácie spočíva vo vylepšovaní algoritmu na generovanie jedálničkov. Ako sme spomínali, generovanie jedálnička obecne je NP-úplný problém, no existuje veľa rôznych algoritmov, ktoré tento problém riešia suboptimálne. V článku z ktorého sme budovali matematický model sa napríklad rozoberá spôsob generovania jedálnička pomocou evolučných algoritmov. Použitie umelej inteligencie určite dokáže túto prácu v budúcnosti rozšíriť veľmi signifikantným spôsobom.

Zoznam použitej literatúry

[1] Carrera-Bastos, Pedro & Fontes-Villalba, Maelán & O'Keefe, James & Lindeberg, Staffan & Cordain, Loren. (2011). The Western diet and lifestyle and diseases of civilization. Research Reports in Clinical Cardiology. 2. 2-15. 10.2147/RRCC.S16919.

[2] GAREY, Michael R.; JOHNSON, David S. *Computers and intractability*. San Francisco: freeman, 1979.

[3] HERNÁNDEZ-OCAÑA, Betania, et al. Bacterial foraging optimization algorithm for menu planning. *IEEE Access*, 2018, 6: 8619-8629.

[4] R. A. Roth, *Nutrition & Diet Therapy,* 10th ed. Mason, OH, USA: Cengage Learning, 2010 *In* HERNÁNDEZ-OCAÑA, Betania, et al. Bacterial foraging optimization algorithm for menu planning. *IEEE Access*, 2018, 6: 8619-8629.

[5] O. P. Perera, "Manual de lineamientos para la práctica de la nutrición clýnica," in *Spanish: Manual of Guidelines for the Practice of Clinical Nutrition*, 1st ed. New York, NY, USA: McGraw-Hill, 2005 *In* HERNÁNDEZ-OCAÑA, Betania, et al. Bacterial foraging optimization algorithm for menu planning. *IEEE Access*, 2018, 6: 8619-8629.

[6] M. Muñoz de Chávez *et al*., "Miriam Muñoz de Chávez," in *Valor nutritivo de los alimentos de mayor consumo*, 2nd ed. New York, NY, USA: McGraw-Hill, 2010 *In* HERNÁNDEZ-OCAÑA, Betania, et al. Bacterial foraging optimization algorithm for menu planning. *IEEE Access*, 2018, 6: 8619-8629.